#153394
SPACE-BASED RADAR PROJE
DREO 3702-194 (SBR)
Copy # 7 of 11 Copies
4 JAN 94
SBR # 826
DREO DRP

UNCLASSIFIED

CLASSIFICATION

National Defence Défense nationale

DEPARTMENT OF NATIONAL DEFENCE

SPACE-BASED RADAR PROJECT

CONTRACTOR REPORT

TITLE: TESTBED DETAILED DESIGN

CONTRACTOR: MacDonald Dettwiler

Raytheon Canada

AUTHOR(S): R. Sawatsky
S. Tran

M. Baskin
R. Collyer
T. Nohara
G. Price
P. Weber

CONTRACTOR'S DOCUMENT NO.: SB-DS-50-4059, Issue 1/0

DATE: September 30, 1993

CONTRACT TITLE:	Development of a Signal Processing Demonstrator System for Space-Based Radar
SSC CONTRACT/FILE NO.:	W8477-0-TD08/01-SV
DND/SBR FILE NO.:	DREO 3702-194 (SBR)
SCIENTIFIC AUTHORITY:	B.C. Eatock

© HER MAJESTY THE QUEEN IN RIGHT OF CANADA (1993)
as represented by the Minister of National Defence

The findings and conclusions presented in this report represent the views of the contractor or author(s). Publication of this report does not constitute DND approval of the contents.

CLASSIFICATION

UNCLASSIFIED

Canada

TESTBED DETAILED DESIGN

Approved by:

Project Engineer: P. Lee   Dec. 3, 93
(Signature/Date)

Project Manager: D. Shilton   D. Shilton Dec 3/93

Audited By:

Quality Assurance: S. Jheeta   D.P. Mansell pp. S. Jheeta.
(Signature/Date) 3 Dec 93

THIS PAGE INTENTIONALLY LEFT BLANK

CHANGE RECORD

ISSUE	DATE	PAGE	DESCRIPTION OF CHANGE	RELEASE
1/0	30/09/93	All	First Revision	RELEASED BY CONFIGURATION CONTROL JBurgess 3 Dec. 93

THIS PAGE INTENTIONALLY LEFT BLANK

50-4059

TABLE OF CONTENTS

Page

SECTION ONE SCOPE 1-1

1.1 IDENTIFICATION 1-1
1.2 SUBSYSTEM OVERVIEW 1-1

1.2.1 Testbed Subsystem 1-1
1.2.2 Relationship to the DSP 1-1

1.3 DOCUMENT OVERVIEW 1-1
1.4 RELATIONSHIP TO OTHER DOCUMENTS 1-3

SECTION TWO DOCUMENTS 2-1

2.1 APPLICABLE DOCUMENTS 2-1
2.2 REFERENCE DOCUMENTS 2-1

SECTION THREE CONTROLLER CSC DETAILED DESIGN 3-1

3.1 SOFTWARE OVERVIEW 3-1
3.2 SOFTWARE DESIGN 3-3

3.2.1 Signal Generation Definition (SGD) CSC 3-3
3.2.2 Simulation Run Definition (SRD) CSC 3-5
3.2.3 RSS Interface (RIF) CSC 3-6
3.2.4 DSP Interface (DIF) CSC 3-7
3.2.5 Data Analysis (DAN) CSC 3-8
3.2.6 Data Management (DMT) CSC 3-9
3.2.7 System Monitor and Control (SMC) CSC 3-14
3.2.8 Utilities (UTL) CSC 3-18

SECTION FOUR RADAR SIGNAL SIMULATOR CSC DETAILED DESIGN 4-1

4.1 SOFTWARE OVERVIEW 4-1

- 4.1.1 Signal Generation Manager CSC 4-4
- 4.1.2 Burst Initialization CSC 4-5
- 4.1.3 Burst Modelling CSC 4-6
- 4.1.4 Playback Manager CSC (PMT) 4-7
- 4.1.5 DSP Interface CSC (DIT) 4-8

SECTION FIVE CONTROLLER WORKSTATION HWCI DETAILED DESIGN 5-1

5.1 HARDWARE OVERVIEW 5-1
5.2 HARDWARE DESIGN 5-3

- 5.2.1 SPARCstation 10 5-3
- 5.2.2 Keyboard 5-4
- 5.2.3 Video Display 5-5
- 5.2.4 Mouse 5-5
- 5.2.5 Speakerbox and Microphone 5-5
- 5.2.6 150 MB Tape Drive 5-6
- 5.2.7 5 GB Tape Drive 5-6
- 5.2.8 CD Drive 5-7
- 5.2.9 Disk 5-7
- 5.2.10 Printer 5-7
- 5.2.11 ADS 5-8
- 5.2.12 Interchangeability (All Major Components) 5-8

5.3 TRANSPORTABILITY, PACKAGING, HANDLING 5-8

- 5.3.1 Transportability 5-8
- 5.3.2 Packaging 5-8
- 5.3.3 Handling 5-9

5.4 SUPPORT EQUIPMENT REQUIREMENTS 5-9

- 5.4.1 Development and Test 5-9
- 5.4.2 Installation 5-9
- 5.4.3 Maintenance 5-9

SECTION SIX RSS WORKSTATION HWCI DETAILED DESIGN 6-1

6.1 HARDWARE OVERVIEW 6-1
6.2 HARDWARE DESIGN 6-3

- 6.2.1 SPARCstation 10 6-3
- 6.2.2 Keyboard 6-3
- 6.2.3 Video Display 6-4
- 6.2.4 Mouse 6-4
- 6.2.5 Speakerbox and Microphone 6-4
- 6.2.6 5 GB Tape Drive 6-5
- 6.2.7 CD Drive 6-5
- 6.2.8 Disk 6-5

6.2.9 ADS 6-6
6.2.10 Vector Co-processor 6-6
6.2.11 Interchangeability (All Major Components) 6-6

6.3 TRANSPORTABILITY, PACKAGING, HANDLING 6-7

6.3.1 Transportability 6-7
6.3.2 Packaging 6-7
6.3.3 Handling 6-7

6.4 SUPPORT EQUIPMENT REQUIREMENTS 6-7

6.4.1 Development and Test 6-7
6.4.2 Installation 6-7
6.4.3 Maintenance 6-7

SECTION SEVEN RTSG HWCI DETAILED DESIGN 7-1

7.1 HARDWARE OVERVIEW 7-1
7.2 HARDWARE DESIGN 7-4

7.2.1 Executive Controller 7-4
7.2.2 Input-Output Processor 7-8
7.2.3 Two Port Memory 7-10
7.2.4 Card Cage 7-12
7.2.5 Backplane 7-14
7.2.6 Power Supply 7-21
7.2.7 Chassis 7-25
7.2.8 Cooling Fans and Filter 7-32
7.2.9 IOP Adapter 7-33

7.3 TRANSPORTABILITY, PACKAGING, HANDLING 7-35

7.3.1 Transportability 7-35
7.3.2 Packaging 7-35
7.3.3 Handling 7-35

7.4 SUPPORT EQUIPMENT REQUIREMENTS 7-35

7.4.1 Development and Test 7-35
7.4.2 Installation 7-36
7.4.3 Maintenance 7-36

SECTION EIGHT RSS/CONTROLLER INTERFACE DETAILED DESIGN 8-1

8.1 INTRODUCTION 8-1
8.2 RSS INTERFACE (RIF) CSC 8-1

8.2.1 RSS Generation Interface (CRGI) CSU 8-3
8.2.2 RSS Playback Interface (CRPI) CSU 8-3
8.2.3 Signal Generation Products (CGEP) CSU 8-4

SECTION NINE REQUIREMENTS TRACEABILITY 9-1

SECTION TEN NOTES 10-1

10.1 ACRONYMS AND ABBREVIATIONS 10-1

APPENDICES

APPENDIX A CONTROLLER CSC DESIGN PROCESS

APPENDIX B RSS CSC DESIGN PROCESS

APPENDIX C DETAILED DESIGN STANDARDS AND CONVENTIONS

APPENDIX D CONTROLLER PDL

APPENDIX E RSS PDL

APPENDIX F RTSG HWCI DESIGN DOCUMENTATION LIBRARY

APPENDIX G HMI DIALOGUE DESIGN

APPENDIX H RTSG BACKPLANE HARDWARE DESIGN DOCUMENTATION LIBRARY

APPENDIX I PARAMETER ESTIMATION FUNCTION

APPENDIX J RSS ALGORITHMS

FIGURES

3-1 CSC/CSU Hierarchy Diagram 3-2

3-2 System Services Tasking Diabram 3-21

4-1 RSS CSC Hierarchy Diagram 4-2

4-2 RSS HWCI Block Diagram....................4-3

5-1 Controller Workstation Equipment Diagram 5-2

6-1 RSS Workstation HWCI Block Diagram 6-2

7-1 RSTG Equipment Diagram 7-3

7-2 Card Cage Mechanical Layout 7-13

7-3 Signal Bus Clock Generator Timing 7-18
7-4 VME Bus Timer with: a) No Timeout and b) Timeout 7-20
7-5 Power Supply Configuration 7-23
7-6 Chassis Bezel Front View 7-27
7-7 Chassis Top View 7-28
7-8 Chassis Side View 7-29
7-9 Rear Panel Connector Area 7-30
8-1 RSS/Controller Interface Diagram 8-2

TABLES

9-1 Testbed Software Requirements Traceability 9-2
9-2 Controller Software Requirements Traceability 9-6
9-3 RSS Software Requirements Traceability 9-8
9-4 DSP/Testbed Interface Software Requirements Traceability 9-9
9-5 Testbed Hardware Requirements Traceability 9-10
9-6 Controller Hardware Requirements Traceability 9-12
9-7 RSS Hardware Requirements Traceability 9-13
9-8 DSP/Testbed Interface Hardware Requirements Traceability 9-14

SECTION ONE

SCOPE

1.1 IDENTIFICATION

This document defines the detailed design of the Space-Based Radar (SBR) Signal Processing Demonstrator (SPD) Testbed subsystem. The design described within this document satisfies all requirements (hardware and software) as defined in the Testbed Specification (Document A–2).

1.2 SUBSYSTEM OVERVIEW

1.2.1 Testbed Subsystem

The Testbed subsystem provides the environment required to operate, test and demonstrate the DSP subsystem. It consists of a Radar Signal Simulator (RSS) and a Testbed Controller functional area. In principle, the Testbed is a complete test instrument capable of being used in conjunction with DSPs other than that developed under this contract.

1.2.2 Relationship to the DSP

The DSP is controlled and operated entirely through an interface to the Testbed subsystem (the "DSP/Testbed Interface").

The Testbed provides digital radar signal data to the DSP, captures radar detection data from the DSP, and provides all DSP command and control. The Testbed provides a Human-Machine Interface (HMI) for overall control of the SPD.

1.3 DOCUMENT OVERVIEW

This design document describes the detailed design of the Testbed Computer Software Components (CSCs) and the major hardware components. For software it describes the CSCs as consisting of Computer Software Units (CSUs) and for hardware it describes the major hardware components.

The organization of this document takes into account the requirement to partition the Testbed into the Testbed Controller and RSS functional areas and also to cover both hardware and software design. These requirements are reflected in the organization of this document, which is as follows:

- Section 1 introduces the Testbed subsystem and this document.
- Section 2 lists the specifications, standards and other reference documents which are wholly or partially applicable to this document.
- Section 3 contains the detailed software design of the Testbed Controller functional area.
- Section 4 contains the detailed software design of the Testbed RSS functional area.
- Section 5 contains the detailed design of the Testbed Controller workstation hardware.
- Section 6 contains the detailed design of the RSS workstation hardware.
- Section 7 contains the detailed design of the Real-Time Signal Generator (RTSG) hardware.
- Section 8 contains the detailed design of the RSS/Controller interface.
- Section 9 contains the requirements traceability showing traceability of Testbed requirements to the hardware and software components.
- Section 10 contains a list of acronyms and abbreviations used in the detailed design documentation.
- Appendix A contains the Testbed Controller detailed design information specific to the design methodology utilized.
- Appendix B contains the RSS detailed design information specific to the design methodology utilized.
- Appendix C contains the Program Design Language (PDL) file standards and conventions used in generating the PDL design information.
- Appendix D contains a listing of the PDL for the Testbed Controller software.
- Appendix E contains a listing of the PDL for the RSS software.
- Appendix F contains the schematics, drawings, parts lists, and other pertinent hardware detailed design information for the Testbed hardware components.
- Appendix G contains the HMI dialogue design.
- Appendix H contains documentation of the RTSG backplane hardware design.
- Appendix I describes the detailed design of the Parameter Estimation function in the data analysis package.

- Appendix J contains updates to the RSS algorithms previously presented in the Testbed Preliminary Design document.

1.4 RELATIONSHIP TO OTHER DOCUMENTS

This detailed design document continues the Testbed hardware and software design process, as documented in the Preliminary Design Document (R-3), but to the next level of design detail.

The software detailed design will decompose each CSC, as identified in the Preliminary Design Document (R-3), into CSUs. This includes allocating the data and processing characteristics for each CSC into its constituent CSUs and further defining the interfaces between the CSUs.

The hardware detailed design will provide more detailed decomposition of the major hardware developmental items as identified in the Preliminary Design Document (R-3). For the major hardware Commercial Off-The-Shelf (COTS) (non-developmental) items, the detailed design information provided will only update any changes to the information provided in the Preliminary Design Document (R-3).

In general, this detailed design document will not duplicate the design information provided in the Preliminary Design Document (R-3), other than to provide a brief introduction to the higher level design components and also to identify where changes have been made to the preliminary design information.

SECTION TWO

DOCUMENTS

2.1 APPLICABLE DOCUMENTS

Ref.	Document	No. Title
A-1	SB-IS-50-4050	DSP/Testbed Interface Specification, MacDonald Dettwiler/Raytheon Canada Ltd., Issue/Revision 2/1, June 1993.
A-2	SB-SY-50-4049	Testbed Specification, MacDonald Dettwiler/Raytheon Canada Ltd., Issue/Revision 2/1, June 1993.

2.2 REFERENCE DOCUMENTS

Ref.	Document	No. Title
R-1	SB-DS-50-4060	DSP/Testbed Interface Detailed Design, MacDonald Dettwiler/Raytheon Canada Ltd., Issue/Revision 1/0, September, 1993.
R-2	SB-DS-50-4058	DSP Detailed Design, MacDonald Dettwiler/Raytheon Canada Ltd., Issue/Revision 1/0, September, 1993.
R-3	SB-DS-50-4054	Testbed Preliminary Design, MacDonald Dettwiler/Raytheon Canada Ltd., Issue/Revision 1/0, March, 1993.
R-4	IR-DD-50-5202	VASP Driver Software Design Document, MacDonald Dettwiler, Issue/Revision 1/0, June 07, 1993.

R-5	X3T9.2/82-2 - Revision 14	SCSI Small Computer System Interface, Computer and Business Equipment Manufacturer's Association.
R-6	MVME712A/D1	MVME712A/MVME712AM/MVME712B Transition Module and MVME147P2 Adapter Board User's Manual, Motorola, April 1989.
R-7	MVME167/D1	MVME167 Single Board Computer User's Manual, Motorola, July 1991.
R-8	ANSI/IEEE	Std 1014-1987 Versatile Backplane Bus (VMEbus Specification), ANSI/IEEE.
R-9		MegaPAC Mechanical and Connection Diagram.
R-10		VASP Technical Manual, consisting of the following documents:
	500-00079	Versatile Array Signal Processor User's Manual
	500-00073	VASP Input/Output Processor Technical Reference Manual
	500-00074	VASP General Signal Processor Technical Reference Manual
	500-00077	VASP Processing Element Executive Technical Reference Manual
	500-00080	VASP Hardware Interface Specifications
	500-00078	VASP Design Examples and Benchmarks
		Spectrum Signal Processing Inc., Revision 1.00, November 1992.
R-11		Digital Signal Processor (DSP) Application Development System (ADS) Reference Manual, Rev. 5.00, 1993. Motorola.
R-12	SB-SY-50-4049	Testbed Specification, MacDonald Dettwiler/Raytheon Canada, Issue/Revision 2/0, March 1993.
R-13	MVME167BUG/D	MVME167 Debugger/Monitor Manual. Motorola.
R-14	ISBN 0-13-640673-4	OSF/Motif Programmer's Guide, Revision 1.1.
R-15	ISBN 0-13-640681-5	OSF/Motif Programmer's Reference, Revision 1.1.
R-16	ISBN 0-13-640616-5	OSF/Motif Style Guide, Revision 1.1.
R-17	ISBN 0-13-043530-9	Application Environment Specification (AES) User Environment Volume, Revision B.

SECTION THREE

CONTROLLER CSC DETAILED DESIGN

3.1 SOFTWARE OVERVIEW

This section provides CSC/CSU hierarchy, CSU high level design and requirements allocation. Detailed design process, design logic and interfaces are provided in Appendices A and D.

The Controller CSC software architecture is shown by the CSC/CSU hierarchy diagram in Figure 3-1. The CSCs and sub-level CSCs are each identified by a three character mnemonic (in brackets) and the CSUs are identified by a four character mnemonic. This is the same CSC hierarchy as provided in Figure 3-2 of the Testbed Preliminary Design document (R-3) with the exception the RSS Generation (RGI) and RSS Playback (RPI) sub-level CSCs have now each been converted from a CSC to a CSU (i.e., CRGI and CRPI CSUs).

The design descriptions, including requirements allocation, for the CSCs identified in the CSC hierarchy diagram, are described under Subsection 3.2.1 of the Preliminary Design Document (R-3) and will not be repeated in detail in the following software design sections. The CSC sections will only provide a brief description of the CSC purpose and a description of any changes from the Preliminary Design Document (Document R-3].

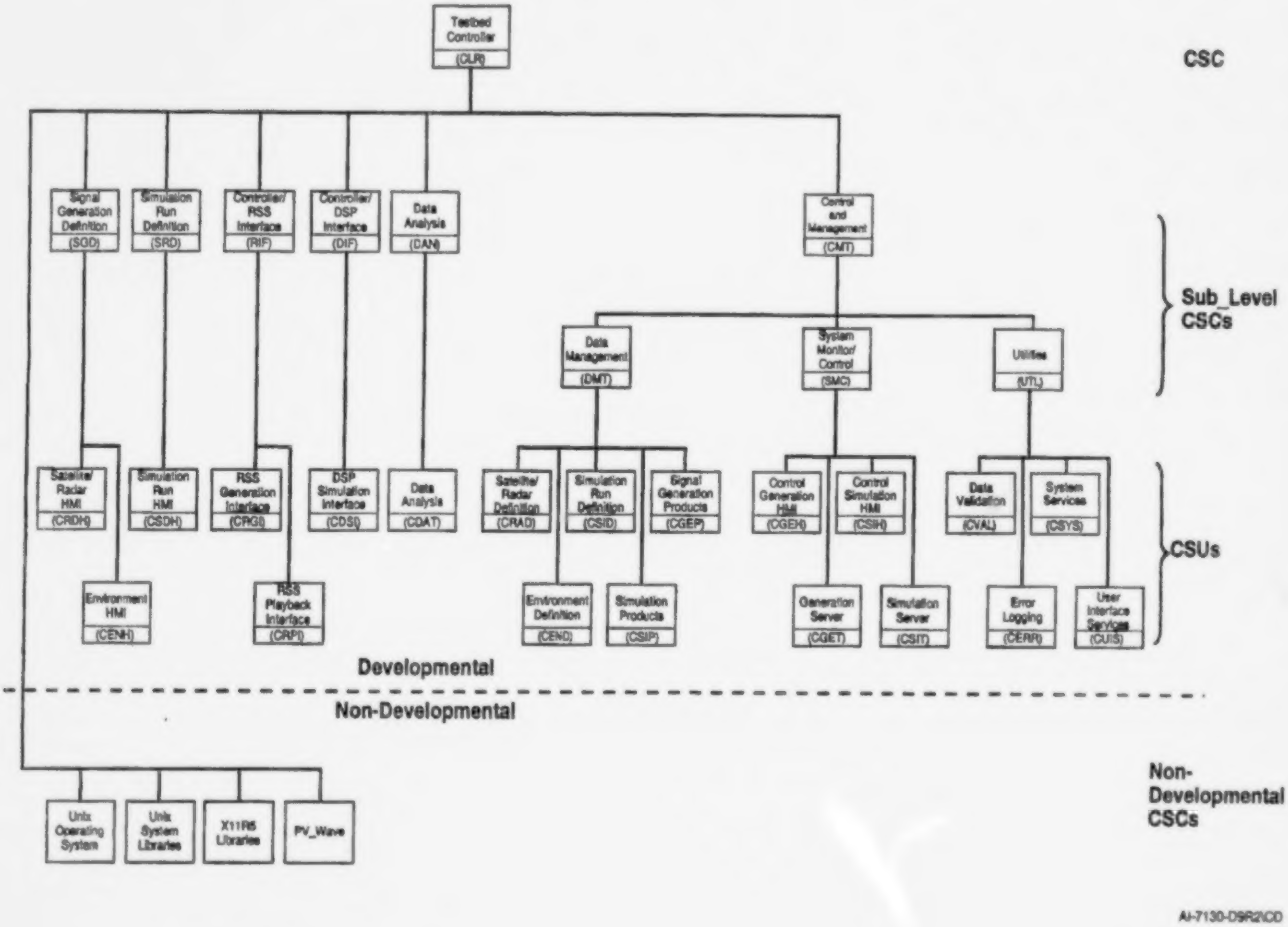

Testbed Controller (CLR)
CSC
Signal Generation Definition (SGD)
Simulation Run Definition (SRD)
Controller/ RSS Interface (RIF)
Controller/ DSP Interface (DIF)
Data Analysis (DAN)
Control and Management (CMT)
Sub_Level CSCs
Data Management (DMT)
System Monitor/ Control (SMC)
Utilities (UTL)
Satellite/ Radar HMI (CRDH)
Environment HMI (CENH)
Simulation Run HMI (CSDH)
RSS Generation Interface (CRGI)
RSS Playback Interface (CRPI)
DSP Simulation Interface (CDSI)
Data Analysis (CDAT)
Satellite/ Radar Definition (CRAD)
Simulation Run Definition (CSID)
Signal Generation Products (CGEP)
Environment Definition (CEND)
Simulation Products (CSIP)
Control Generation HMI (CGEH)
Control Simulation HMI (CSIH)
Generation Server (CGET)
Simulation Server (CSIT)
Data Validation (CVAL)
System Services (CSYS)
Error Logging (CERR)
User Interface Services (CUIS)
CSUs
Developmental
Non-Developmental
Unix Operating System
Unix System Libraries
X11R5 Libraries
PV_Wave
Non-Developmental CSCs
AI-7130-D9R2\CD

Figure 3-1 CSC/CSU Hierarchy Diagram

3.2 SOFTWARE DESIGN

3.2.1 Signal Generation Definition (SGD) CSC

The SGD CSC provides the functionality required for the operator to define the signal generation parameters to be used by the RSS CSC in generating the Simulated Radar Signal Data (SRSD) file and optional intermediate products. This functionality includes the HMI to define the satellite/radar and environment related parameters. Refer to Preliminary Design Document (Document R-3) Subsection 3.2.1.1 for a more detailed description of the SGD CSC.

The SGD CSC consists of the following CSUs:

- Controller: Satellite/Radar HMI (CRDH)
- Controller: Environment HMI (CENH)

3.2.1.1 CRDH CSU

The purpose of the CRDH CSU is to provide the functionality for the operator to enter the satellite and radar related parameters as specified in Appendix A of the Testbed Specification (Document A-2). This functionality includes providing control, sequencing and manipulation of HMI dialogues for entry of logically related parameter groups (structural dependencies), default provision and validation processing.

3.2.1.1.1 Requirements

The following requirements have been allocated from the Testbed Specification (Document A-2) to this CSU:

Number	Title
3.1.2.0.1	External Events Recognized
3.2.1.0.1	Define Radar Signal Generation Response
3.3.1.0.1	Multiple User Operation
3.3.1.0.2	Operation Without DSP
3.3.2.0.1	Define Signal Generation Parameters
3.3.2.0.6	Baseline Definition
3.4.1.1.2	Operator Multitasking
3.4.1.1.3	Default Values
3.4.1.1.4	Units of Measure
3.4.1.1.5	Positive Confirmation
3.4.1.2.1	Signal Generation Parameter Hierarchy
3.4.1.2.3	Data Segment Interval
3.4.1.2.4	CT Interval
3.4.1.2.5	Dwell Interval (DI)
3.4.1.2.6	Non-Coherent Interval (NCI) Interval
3.4.1.2.7	Coherent Porcessing (CP) Interval
3.4.1.2.9	Data Capture Options
3.4.1.2.10	Signal Generation Data Management
3.4.1.4.1	Signal Generation Model Management
3.4.1.7.3	Offline Access

4.1.1.1.1	Operation Capabilities
4.1.1.1.2	HMI Capabilities
4.1.1.2.1	HMI for Signal Generation Definition
4.1.1.4.1	HMI for Signal Generation Models
4.1.1.7.5	Maintain Errorlog
4.2.1.1.1	Controller/HMI Software Interface

3.2.1.1.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of loc data files or data base, and limitations for use of each CSU are explained as applicable in th following sections:

a) Section A6.1 - provides a description of the Satellite/Radar Definition task, including task structure and external interfaces;

b) Section D1.1 - provides a complete description of the interface specification and detailed PDL logic;

c) Section G2 - provides a description of the HMI dialogue design for the user interface.

3.2.1.2 CENH CSU

The purpose of the CENH CSU is to provide the functionality for the operator to enter the environment related parameters as specified in Appendix A of the Testbed specification (A-2). This functionality includes providing control and sequencing of HMI dialogues for ent of logically related parameter groups (structural dependencies), default provision and validation processing.

3.2.1.2.1 Requirements

The following requirements have been allocated from the Testbed Specification (A-2) to thi CSU:

Number	Title
3.1.2.0.1	External Events Recognized
3.2.1.0.1	Define Radar Signal Generation Response
3.3.1.0.1	Multiple User Operation
3.3.1.0.2	Operation Without DSP
3.3.2.0.1	Define Signal Generation Parameters
3.4.1.1.2	Operator Multitasking
3.4.1.1.3	Default Values
3.4.1.1.4	Units of Measure
3.4.1.1.5	Positive Confirmation
3.4.1.2.8	Environment Definition

3.4.1.2.10	Signal Generation Data Management
3.4.1.7.3	Offline Access
4.1.1.1.1	Operation Capabilities
4.1.1.1.2	HMI Capabilities
4.1.1.2.1	HMI for Signal Generation Definition
4.1.1.7.5	Maintain Errorlog
4.2.1.1.1	Controller/HMI Software Interface

3.2.1.2.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of local data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A6.2 - provides a description of the Environment Definition task, including task structure and external interfaces;

b) Section D1.2 - provides a complete description of the interface specification and detailed PDL logic;

c) Section G3 - provides a description of the HMI dialogue design for the user interface.

3.2.2 Simulation Run Definition (SRD) CSC

The SRD CSC provides the functionality required for the operator to define the simulation run parameters to be used by the RSS CSC in playback of the SRSD file and by the DSP to control the processing of the input signal data and generate the detection and optional intermediate product data.

The SRD CSC consists of the following CSU:

- Controller: Simulation Run HMI (CSDH)

3.2.2.1 CSDH CSU

The purpose of the CSDH CSU is to provide the functionality for the operator to enter the simulation run related parameters as specified in Appendix A of the Testbed specification (Document A-2). This functionality includes providing control, sequencing and manipulation of HMI dialogues for entry of logically related parameter groups (structural dependencies), default provision and validation processing.

3.2.2.1.1 Requirements

The following requirements have been allocated from the Testbed Specification (Document A–2) to this CSU:

Number	Title
3.1.2.0.1	External Events Recognized
3.2.1.0.3	Define Simulation Run Response
3.3.1.0.1	Multiple User Operation
3.3.1.0.2	Operation Without DSP
3.3.2.0.3	Define Simulation Run Parameters
3.3.2.0.6	Baseline Definition
3.3.2.0.7	Signal Processing Reconfigurability Definition
3.4.1.1.2	Operator Multitasking
3.4.1.1.3	Default Values
3.4.1.1.4	Units of Measure
3.4.1.1.5	Positive Confirmation
3.4.1.3.1	Simulation Run Parameter Hierarchy
3.4.1.3.2	Data Segment Identifier (ID)
3.4.1.3.3	Radar Signal Processor (RSP) Configuration
3.4.1.3.6	ESP Configuration
3.4.1.3.7	DSP Processing Parameters
3.4.1.3.8	Data Capture Options
3.4.1.3.9	Simulation Run Data Management
3.4.1.7.3	Offline Access
3.6.1.2.1	Software Reconfigurability
4.1.1.1.1	Operation Capabilities
4.1.1.1.2	HMI Capabilities
4.1.1.3.1	HMI for Simulation Run Definition
4.1.1.7.5	Maintain Errorlog
4.2.1.1.1	Controller/HMI Software Interface

3.2.2.1.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of loc data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A6.3 - provides a description of the Simulation Definition task, including task structure and external interfaces;

b) Section D1.3 - provides a complete description of the interface specification and detailed PDL logic;

c) Section G4 - provides a description of the HMI dialogue design for the user interface.

3.2.3 RSS Interface (RIF) CSC

The RIF CSC provides the definition of the interface between the Controller and the RSS CSCs for signal generation and data playback activities. The high level design and requirements allocation for this CSC and its constituent CSUs are described in Section 8 where the RSS/Controller Interface design information is captured.

3.2.4 DSP Interface (DIF) CSC

The DIF CSC provides the definition of the interface between the Controller and the DSP (ESP and RSP) CSCs for a simulation run activity. This defines all the messages, message content and format and the protocol used in sending messages between the two CSCs. This does not include any application software to send and receive the messages.

The DIF CSC described in this detailed design document differs from the DIF CSC described in the preliminary design document (R-1) in the following areas:

a) the DIF CSC no longer includes the Simulation Server task to communicate with the DSP (ESP and RSP executive controllers). This has now been allocated, along with the corresponding requirements, as part of the System Monitor and Control (SMC) CSC.

The DIF CSC consists of the following CSU:

- Controller: DSP Simulation Interface (CDSI) CSU

3.2.4.1 CDSI CSU

The purpose of the CDSI CSU is to provide the definition of the interface between the Controller and DSP for simulation run activities. This defines all the messages, message content and format and the protocol used in sending messages.

3.2.4.1.1 Requirements

The following requirements have been allocated from the DSP/Testbed Interface Specification (A-1) to this CSU:

Number	Title
3.2.2.1.1	Concurrency
3.2.2.1.2	Sources and Sinks
3.2.2.1.3	Application Protocol
3.2.2.1.4	Preconditions
3.2.2.2.1	Detection Data Element Definition
3.2.2.2.2	Intermediate Data Element Definition
3.2.3.1.1	Concurrency
3.2.3.1.2	Sources and Sinks
3.2.3.1.3	Application Protocol
3.2.3.2.1	DSP Commands

The following requirements have been allocated from the Testbed specification (Document A–2) to this CSU:

Number	Title
3.7.1.1.1	Interface Requirements
4.1.1.9.1	Interface Control
4.2.2.1.1	Controller/RSS Software Interface

3.2.4.1.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of loc data files or data base, and limitations for use of each CSU are explained as applicable in the following:

a) DSP/Testbed Interface Detailed Design document (R-1) - provides a complete description of the interface specification.

3.2.5 Data Analysis (DAN) CSC

The DAN CSC provides the functionality for the operator to select the desired product files and perform data management operations or perform further data reduction and analysis functions.

The DAN CSC consists of the following CSU:

- Controller: Data Analysis (CDAT) CSU

3.2.5.1 Data Analysis (CDAT) CSU

The purpose of the CDAT CSU is to provide the functionality for the operator to perform further data analysis operations on intermediate products captured from the activity of generating the signal data, or from intermediate and detection products captured from an actual simulation run. This functionality includes providing the user with a broad range of tools to manipulate and display the data and to further develop customized analysis function

3.2.5.1.1 Requirements

The following requirements have been allocated from the Testbed Specification (Document A–2) to this CSU:

Number	Title
3.1.2.0.1	External Events Recognized
3.2.1.0.6	Analyze Results Response
3.3.1.0.1	Multiple User Operation
3.3.1.0.2	Operation Without DSP
3.3.4.0.1	Analysis Capability

3.3.4.0.2	Hard Copy Output
3.3.4.0.3	Parameter Estimation
3.4.1.1.2	Operator Multitasking
3.4.1.8.1	Select Data Sets
4.1.1.1.1	Operational Capabilities
4.1.1.1.2	HMI Capabilities
4.1.1.8.1	HMI For Analyze Simulation Results
4.1.1.8.2	Analysis Capability
4.2.2.1.1	Controller/RSS Software Interface

3.2.5.1.2 Design Description

The CDAT CSU Data Analysis task will be provided entirely by the COTS PV-Wave Point and Click to provide the user interface and by the PV-Wave Command Language to provide the customized data analysis procedures. Also provided will be a standard set of Command Language procedures to be used to ingest each of the product types as generated by the signal generation and simulation run activities.

For further design information, refer to the following sections:

a) Section A6.8 - provides a description of the Data Analysis task, specifically the external data interfaces;

b) Section D1.7 - provides a description of the PV-Wave Command Language procedures to be used for ingest of data into PV-Wave as well as for Parameter Estimation of RSP detection data;

c) PV-Wave documentation set for any further design details.

3.2.6 Data Management (DMT) CSC

The DMT CSC provides the definition of the data stores that are shared throughout the Testbed Controller CSC. Definitions are provided for the following data stores:

a) Satellite/Radar Definition
b) Environment Definition
c) Simulation Run Definition
d) Simulation Run Products
e) Signal Generation Products

The DMT CSC described in this detailed design document differs from the DMT CSC described in the preliminary design document (Document R-3) in the following areas:

a) the errorlog files now are included in the Utilities (UTL) CSC as part of the error logging utilities;

b) the Signal Generation Definition is now decomposed into separate Satellite/Radar and Environment Definitions;

c) the Signal Generation Status and Simulation Run Status files have been eliminated along with batching of generation and simulation activities.

The DMT CSC consists of the following CSUs:

- Controller: Satellite/Radar Definition (CRAD) CSU
- Controller: Environment Definition (CEND) CSU
- Controller: Simulation Run Definition (CSID) CSU
- Controller: Simulation Run Products (CSIP) CSU
- Controller: Signal Generation Products (CGEP) CSU

3.2.6.1 CRAD CSU

The purpose of the CRAD CSU is to provide the definition of the Satellite/Radar Definition data store as well as the data file read and write access functions. This consists of all the satellite and radar related parameters input as part of the Satellite/Radar HMI dialogues defined in Appendix G.

3.2.6.1.1 Requirements

No requirements have been allocated from the Testbed specification (Document A-2) or DSP/Testbed interface specification (Document A-1) to this CSU.

3.2.6.1.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of loc data files or data base, and limitations for use of each CSU are explained as applicable in th following sections:

a) Section A5.2.3 - provides a description of the file characteristics;

b) Section D1.8 - provides a complete description of the Satellite/Radar Definition data structure and the data file read and write service functions.

3.2.6.2 CEND CSU

The purpose of the CEND CSU is to provide the definition of the Environment Definition data store as well as the data file read and write access functions. This consists of all the environment related parameters input as part of the Environment HMI dialogues defined in Appendix G.

3.2.6.2.1 Requirements

No requirements have been allocated from the Testbed specification (Document A-2) or DSP/Testbed interface specification (Document A-1) to this CSU.

3.2.6.2.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of local data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A5.2.3 - provides a description of the file characteristics;

b) Section D1.9 - provides a complete description of the Environment Definition data structure and the data file read and write service functions.

3.2.6.3 CSID CSU

The purpose of the CSID CSU is to provide the definition of the Simulation Run Definition data store as well as the data file read and write access functions. This consists of all the simulation run related parameters input as part of the Simulation Run HMI dialogues defined in Appendix G.

3.2.6.3.1 Requirements

No requirements have been allocated from the Testbed specification (Document A-2) or DSP/Testbed interface specification (Document A-1) to this CSU.

3.2.6.3.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of local data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A5.2.3 - provides a description of the file characteristics;

b) Section D1.10 - provides a complete description of the Simulation Run Definition data structure and the data file read and write service functions.

3.2.6.4 CSIP CSU

The purpose of the CSIP CSU is to provide the definition of the products generated by the DSP during a simulation run. This consists of a separate definition for each of the following product files:

a) Radar Detections;

b) ESP Intermediate Products as follows:

- ESP input signal
- Post-SC signals
- Post nulling signals
- Post blanking signals
- Post Multiple Antenna Surveillance Radar (MASR) signals
- Post Frequency Alignment (FA) signals
- Post Pulse Doppler (PD) signals
- Nulling weight vectors
- Correlation matrices
- Cholesky-factor matrices
- Adaptive nulling snapshots
- Pulse doppler window function

c) RSP Intermediate Products as follows:

- RSP input signals
- Post Pulse Compression (PC) signals
- Post Clutter Cancellation (CC) signals
- RCS signals
- Blanked signal
- Post NCI signal
- Monopulse angle data
- Arrested Synthetic Aperture Radar (ASAR) Displaced Phase Centre Antenna (DPCA) weights
- Target deweighting function
- Pulse Compression replicas

3.2.6.4.1 Requirements

The following requirements have been allocated from the DSP/Testbed Interface Specificat (Document A-1) to this CSU:

Number	Title
3.2.2.2.1	Detection Data Element Definition
3.2.2.2.2	Intermediate Data Element Definition

The following requirements have been allocated from the Testbed Specification (Documei 2) to this CSU:

Number	Title
3.3.3.2.1	Data Logging
4.1.1.9.2	Data Logging
4.1.1.10.2	Data Logging

3.2.6.4.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of local data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A5.2.3 - provides a description of the file characteristics;

b) DSP/Testbed Interface Detailed Design Document (Document R-1) - provides a complete description of the interface specification.

3.2.6.5 CGEP CSU

The purpose of the CGEP CSU is to provide the definition of the products generated by the RSS during a signal generation activity which will be used by the Controller software during simulation run and data analysis. The high level design and requirements allocation for this CSU are described in Section 8 where the RSS/Controller Interface design information is captured.

3.2.6.5.1 Requirements

The following requirements have been allocated from the Testbed Specification (Document A–2) to this CSU:

Number	Title
3.3.3.1.1	Data Logging
4.1.1.9.2	Data Logging

3.2.6.5.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of local data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A5.2.3 - provides a description of the file characteristics;

b) Section D1.12 - provides a complete description of the interface specification.

3.2.7 System Monitor and Control (SMC) CSC

The SMC CSC provides the functionality to monitor and control the signal generation and simulation run activities. This includes providing the HMI to invoke and control each activi and the interface software to communicate with the RSS and DSP components.

The SMC CSC described in this detailed design document differs from the SMC CSC described in the preliminary design document (Document R-3) in the following areas:

a) the SMC CSC now also includes the Generation Server and Simulation Server tasks to communicate with the DSP (ESP and RSP executive controllers) and the RSS. I the preliminary design, this was allocated to the DIF and RGI CSCs respectively.

The SMC CSC consists of the following CSUs:

- Controller: Control Generation HMI (CGEH) CSU
- Controller: Generation Server (CGET) CSU
- Controller: Control Simulation HMI (CSIH) CSU
- Controller: Simulation Server (CSIT) CSU

3.2.7.1 CGEH CSU

The purpose of the CGEH CSU is to provide the functionality for the operator to invoke an control signal generation activity and monitor the activity progress.

3.2.7.1.1 Requirements

The following requirements have been allocated from the Testbed Specification (Document 2) to this CSU:

Number	Title
3.1.2.0.1	External Events Recognized
3.2.1.0.2	Invoke Radar Signal Generation Response
3.4.1.1.4	Units of Measure
3.4.1.5.1	Signal Generation HMI
3.4.1.7.3	Offline Access
4.1.1.5.1	HMI For Invoke Signal Generation
4.1.1.7.4	Control of Activities
4.1.1.7.5	Maintain Errorlog
4.2.1.1.1	Controller/HMI Software Interface

3.2.7.1.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of local data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A6.4 - provides a description of the Monitor/Control Generation task, including task structure and external interfaces;

b) Section D1.13 - provides a complete description of the interface specification and detailed PDL logic;

c) Section G5 - provides a description of the HMI dialogue design for the user interface.

3.2.7.2 CGET CSU

The purpose of the CGET CSU is to provide the functionality of initiating the signal generation activity as specified by the parameters selected by the operator in the CGEH CSU and providing the interface between the Controller and the RSS for the duration of the generation activity.

3.2.7.2.1 Requirements

The following requirements have been allocated from the Testbed Specification (Document A-2) to this CSU:

Number	Title
3.1.2.0.1	External Events Recognized
3.2.1.0.2	Invoke Radar Signal Generation Response
4.1.1.7.4	Control of Activities
4.1.1.7.5	Maintain Errorlog
4.1.1.9.1	Interface Control

3.2.7.2.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of local data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A6.5 - provides a description of the Generation Server task, including task structure and external interfaces;

b) Section D1.14 - provides a complete description of the interface specification and detailed PDL logic.

3.2.7.3 CSIH CSU

The purpose of the CSIH CSU is to provide the functionality for the operator to invoke and control the simulation run activity and monitor the activity progress.

3.2.7.3.1 Requirements

The following requirements have been allocated from the Testbed Specification (Document 2) to this CSU:

Number	Title
3.1.2.0.1	External Events Recognized
3.2.1.0.4	Invoke Simulation Run Response
3.3.3.2.2	Performance Metrics Display
3.4.1.1.4	Units of Measure
3.4.1.6.1	Simulation Run HMI
3.4.1.7.3	Offline Access
4.1.1.6.1	HMI for Invoke Simulation Run
4.1.1.7.4	Control of Activities
4.1.1.7.5	Maintain Errorlog
4.2.1.1.1	Controller/HMI Software Interface

3.2.7.3.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of loca data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A6.6 - provides a description of the Monitor/Control Simulation task, including task structure and external interfaces;

b) Section D1.15 - provides a complete description of the interface specification and detailed PDL logic;

c) Section G6 - provides a description of the HMI dialogue design for the user interface.

3.2.7.4 CSIT CSU

The purpose of the CSIT CSU is to provide the functionality of initiating the simulation run activity as specified by the parameters selected by the operator in the CSIH CSU and providing the interface between the Controller and both the RSS and DSP for the duration of the simulation activity.

3.2.7.4.1 Requirements

The following requirements have been allocated from the DSP/Testbed Interface Specification (Document A-1) to this CSU:

Number	Title
3.2.2.1.1	Concurrency
3.2.2.1.2	Sources and Sinks
3.2.2.1.3	Application Protocol
3.2.2.1.4	Preconditions
3.2.3.1.1	Concurrency
3.2.3.1.2	Sources and Sinks
3.2.3.1.3	Application Protocol
3.2.3.2.1	DSP Commands

The following requirements have been allocated from the Testbed Specification (Document A-2) to this CSU:

Number	Title
3.1.2.0.1	External Events Recognized
3.2.1.0.4	Invoke Simulation Run Response
3.2.1.0.5	Capture DSP Simulation Run Response
3.4.3.1.1	DSP Software Interface
3.5.1.1.1	Real Time Operation
3.5.1.1.2	Scope of Real-Time Performance Demonstration
4.1.1.7.4	Control of Activities
4.1.1.7.5	Maintain Errorlog
4.1.1.9.1	Interface Control
4.1.1.10.1	Interface Control
4.1.1.10.3	Simulation Metrics
4.2.1.1.1	Controller/HMI Software Interface
4.2.3.0.1	Controller/DSP Interface
4.3.1.0.1	Real Time Demonstration

3.2.7.4.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of local data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A6.7 - provides a description of the Simulation Server task, including task structure and external interfaces;

b) Section D1.16 - provides a complete description of the interface specification and detailed PDL logic.

3.2.8 UTL CSC

The Utilities CSC provides the common functions such as math routines, system services, err handling, data validation and inter-task communication to other CSCs.

The UTL CSC described in this detailed design document differs from the UTL CSC described in the preliminary design document (Document R-3) in the following areas:

a) the UTL CSC now also includes the error handling functions, including definitions of the error log files;

b) the UTL CSC now also includes the data validation functions for "black box" validation of Satellite/Radar, Environment, and Simulation Run definitions;

c) the UTL CSC now also includes startup and shutdown functions for the Generation Server, RSS Generation Manager, Simulation Server and RSS Playback Manager tasks (see Figure A-1);

d) the UTL CSC now also includes common routines for the HMI tasks (Definition a Monitor/Control tasks in Figure A-1).

The UTL CSC consists of the following CSUs:

- Controller: Data Validation (CVAL) CSU
- Controller: Error Logging (CERR) CSU
- Controller: System Services (CSYS) CSU
- Controller: User Interface Services (CUIS) CSU

3.2.8.1 CVAL CSU

The purpose of the CVAL CSU is to provide the functionality of validating the parameter definitions to be used in the signal generation and simulation run activities. This validation function is a "black box" type of validation and as such will validate the entire definition, from top to bottom, ensuring internal dependencies as well as dependencies on related definitions are satisfied. The validation functions provided will include as follows:

a) satellite/radar definition validated for internal dependencies, including RTSG memory limit;

b) environment definition validated for internal dependencies as well as dependencies on satellite/radar definition;

c) simulation run definition validated for internal dependencies as well as dependencies on the Satellite/Radar Definition associated with the RSS derived processing parameters file.

The above validation functions are performed when the operator manually selects the appropriate validation function; or when the operator invokes the execution of signal generation or simulation run activities thus automatically performing the appropriate validation functions.

3.2.8.1.1 Requirements

The following requirements have been allocated from the Testbed Specification (Document A-2) to this CSU:

Number	Title
3.4.1.2.11	Satellite/Radar Parameter Validation
3.4.1.2.12	Environment Parameter Validation
3.4.1.3.10	Simulation Run Parameter Validation
4.1.1.2.2	Signal Generation Validation
4.1.1.3.2	Simulation Run Validation

3.2.8.1.2 Design Description

The input data elements from the selected satellite and radar definition (CRAD CSU) are validated internally, as well as the selected simulation run definition (CSID CSU) with the DSP derived parameters (CGEP CSU) produced during a signal generation run. The capabilities and design description are given in the following sections:

a) Section A6.1 - describes the tasking structure in which the satellite and radar validation operates;

   Section A6.3 - describes the tasking structure in which the simulation run validation operates;

   Section A6.4 - describes the tasking structure in which the monitor and control of a generation run and it's associated validation operate, and;

   Section A6.6 - describes the tasking structure in which the monitor and control of a simulation run and it's associated validation operate.

b) Section D1.17 - provides a complete description of the interfaces and the detailed PDL logic.

CERR CSU

The purpose of the CERR CSU is to provide a set of common services to other Controller CSUs for initializing an error log file and logging error or warning message to the file in a formatted manner.

3.2.8.1.3 Requirements

The following requirements have been allocated from the Testbed Specification (Document 2) to this CSU:

Number	Title
4.1.1.7.5	Maintain Errorlog

3.2.8.1.4 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of loc data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A5.2.3 - provides a description of the file characteristics;

b) Section D1.18 - provides a complete description of the interface specification and detailed PDL logic.

3.2.8.2 CSYS CSU

The purpose of the CSYS CSU is to provide the system services, including services to startup and shutdown the SPD system daemons that execute in the background for control of signal generation and simulation run activities. The system services tasking diagram is shown in Figure 3-2.

Requirements

The following requirements have been allocated from the Testbed Specification (Document A–2) to this CSU:

Number	Title
3.4.1.7.1	SPD User Access
3.4.1.7.2	SPD System Manager Access
4.1.1.7.1	HMI For Perform System Control

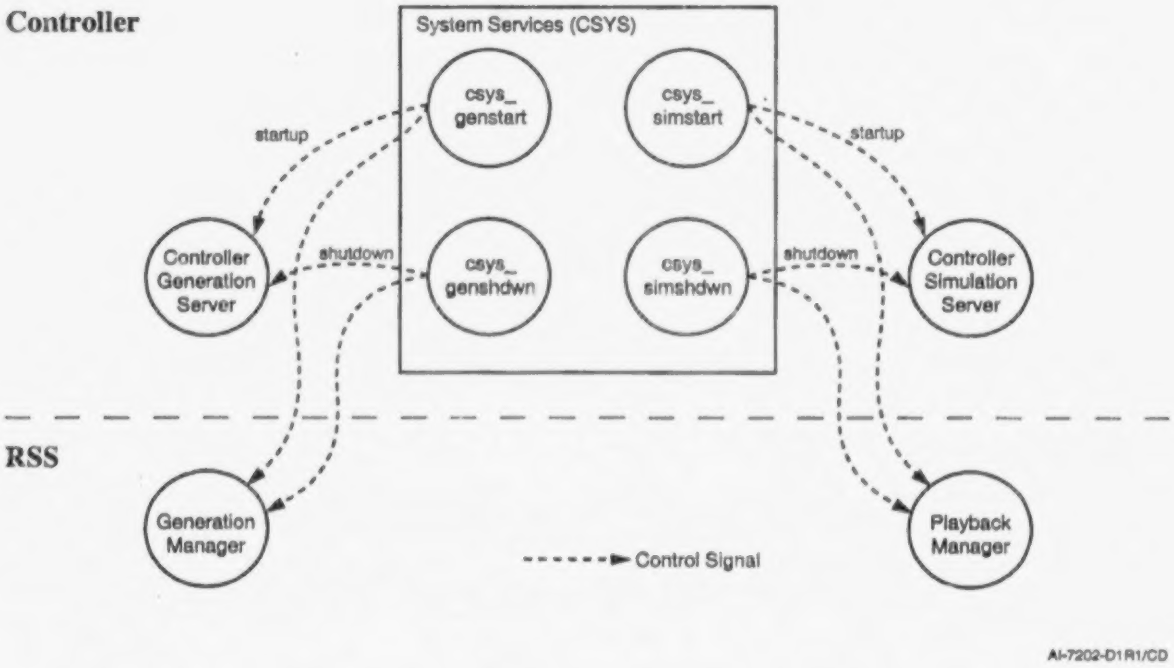

Controller
System Services (CSYS)
csys_ genstart
csys_ simstart
startup
startup
Controller Generation Server
shutdown
csys_ genshdwn
csys_ simshdwn
shutdown
Controller Simulation Server
RSS
Generation Manager
Control Signal
Playback Manager
AI-7202-D1R1/CD

Figure 3-2 System Services Tasking Diagram

3.2.8.3 CSYS CSU

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of loc data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section D1.19 - provides a complete description of the interface specification and detailed PDL logic.

3.2.8.4 CUIS CSU

The purpose of the CUIS CSU is to provide the common User Interface Services (UIS) that are used by the other user interface CSUs such as CRDH, CENH, CSDH, CGEH and CSIH. The user interface services provided include services such as:

a) common HMI dialogues (i.e., confirmation dialogue, warning dialogue, informatic dialogue, file selection dialogue and status dialogue);

b) common HMI related services;

c) interfaces to services within the X window toolkit (Ux).

3.2.8.4.1 Requirements

No requirements have been allocated from the Testbed Specification (Document A-2) or DSP/Testbed Interface Specification (Document A-1) to this CSU.

3.2.8.4.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of loc data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section D1.20 - provides a source listing of the UxXt package element which contains a basic subset of Ux library functions used by UIM/X generated HMI coc This software is extracted from the UIM/X COTS software.

SECTION 4

RSS CSC DETAILED DESIGN

4.0.1 Changes from Preliminary Design Baseline

The RSS CSC has undergone 2 changes from that presented as the preliminary design baseline at PDR:

a) Additional signal generation task to handle RPC server functions, and
b) Addition of Utilities CSC for common functions available to all CSCs.

4.1 SOFTWARE OVERVIEW

The RSS top level CSC is decomposed into the following CSCs:

a) Signal Generation Manager,
b) Burst Initialization,
c) Burst Modelling,
d) Playback Manager,
e) DSP Interface, and
f) Utilities.

The RSS CSC architecture is summarized in the CSC hierarchy diagram in Figure 4-1. The developmental CSCs shown in this hierarchy diagrams are identical with those defined

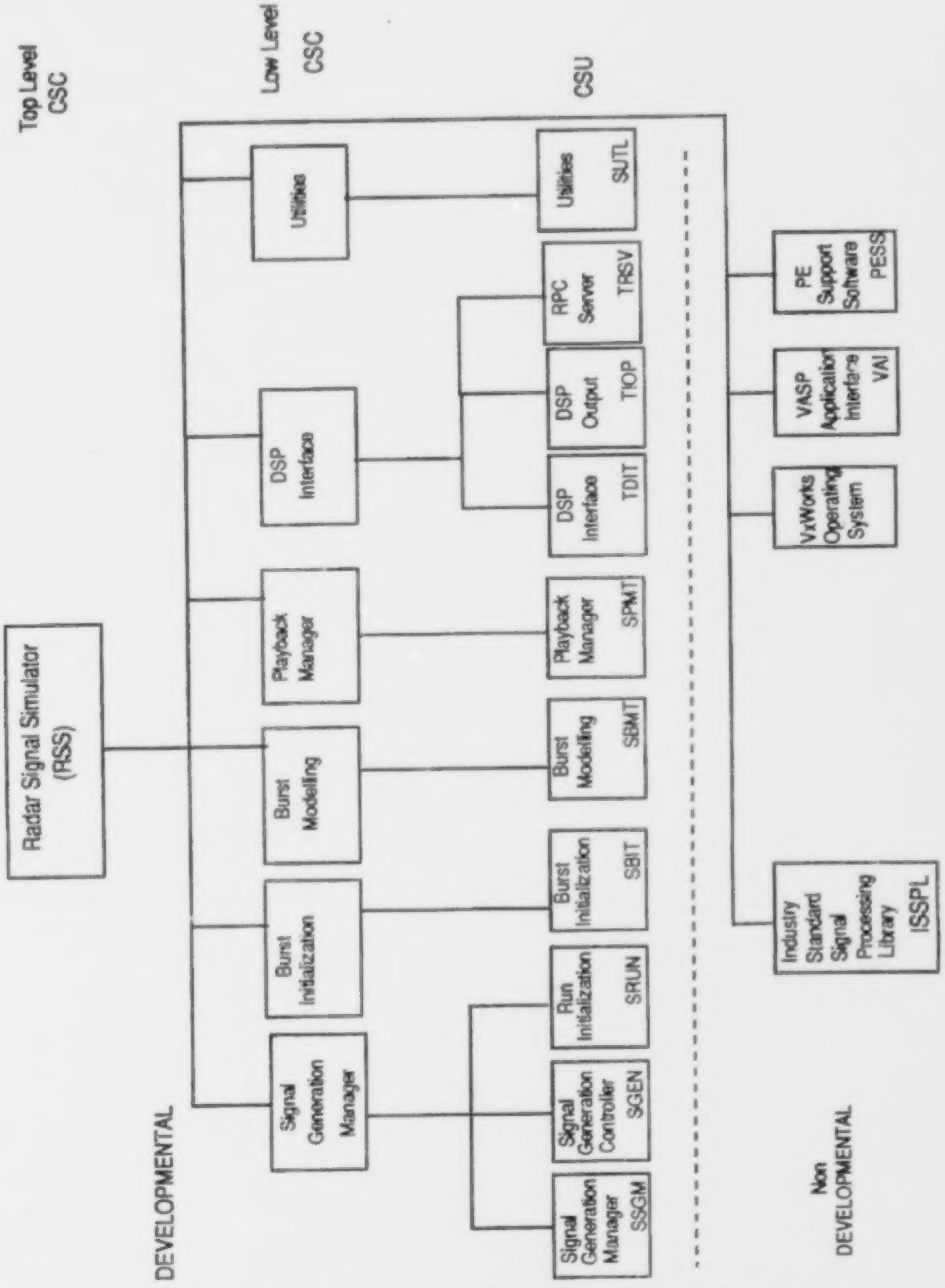

Top Level CSC
Low Level CSC
CSU
Radar Signal Simulator (RSS)
DEVELOPMENTAL
Signal Generation Manager
Burst Initialization
Burst Modelling
Playback Manager
DSP Interface
Utilities
Signal Generation Manager SSGM
Signal Generation Controller SGEN
Run Initialization SRUN
Burst Initialization SBIT
Burst Modelling SBMT
Playback Manager SPMT
DSP Interface TDIT
DSP Output TIOP
RPC Server TRSV
Utilities SUTL
Non DEVELOPMENTAL
Industry Standard Signal Processing Library ISSPL
VxWorks Operating System
VASP Application Interface VAI
PE Support Software PESS

Figure 4-1 RSS CSC HIERARCHY DIAGRAM

in [R-3], but the diagram has been updated to include the non-developmental CSCs as well as the defined CSUs.

The design descriptions, including requirements allocation, for the developmental CSCs identified in the CSC hierarchy diagram, are described in section 3.2.2 of [R-3] and will not be repeated in detail in the following software design sections. The CSC sections will only provide a brief description of the CSC purpose and a description of any changes from the preliminary design in [R-3].

In order to clearly explain the execution environment of these CSCs in the following sections, a block diagram of the RSS Workstation and Real Time Signal Generator (RTSG) HWCIs indicating the allocation of CSCs is shown in Figure 4-2. The characteristics of the RSS workstation HWCI and RTSG HWCI are described in sections 6 and 7, respectively.

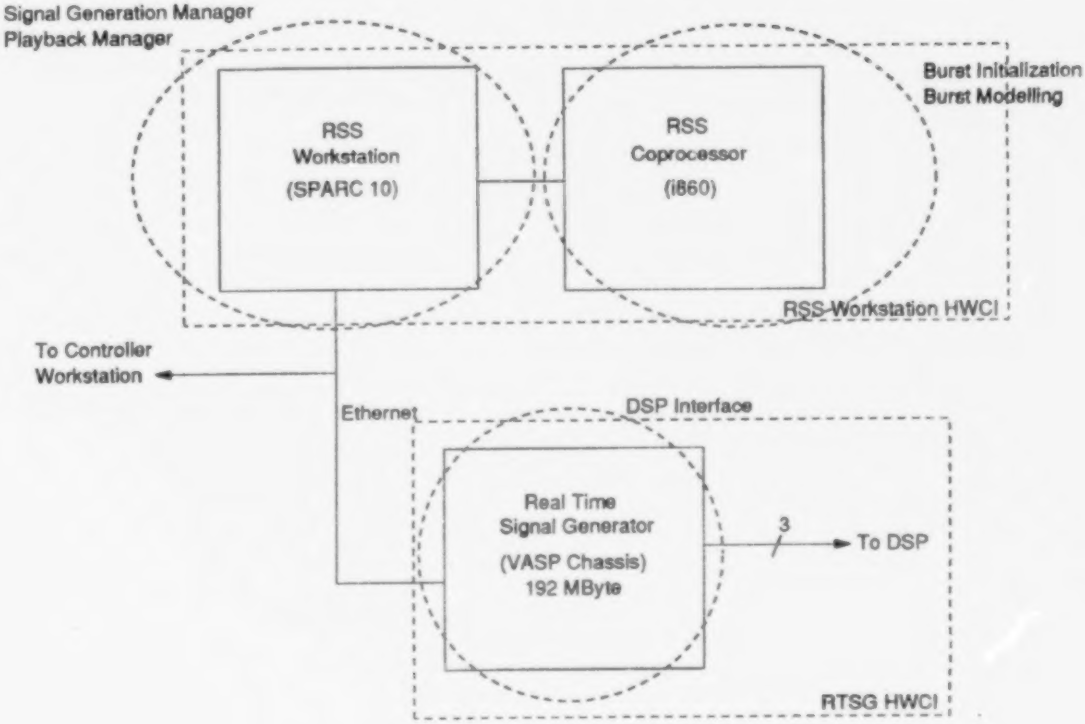

Signal Generation Manager
Playback Manager
RSS
Workstation
(SPARC 10)
RSS
Coprocessor
(i860)
Burst Initialization
Burst Modelling
RSS Workstation HWCI
To Controller
Workstation
Ethernet
DSP Interface
Real Time
Signal Generator
(VASP Chassis)
192 MByte
3
To DSP
RTSG HWCI

Figure 4-2 RSS HWCI BLOCK DIAGRAM

4.1.1 Signal Generation Manager CSC

The Signal Generation Manager CSC is fully decomposed into the following CSUs:

a) Signal Generation Manager (SSGM) CSU,
b) Signal Generation Controller (SGEN) CSU, and
c) Run Initialization (SRUN) CSU.

The Signal Generation Manager CSC executes on the RSS workstation, and provides the functionality to control other tasks associated with the signal generation activity of the RSS, and to communicate with the Testbed Controller functional area.

4.1.1.1 SSGM CSU

The SSGM CSU is the primary signal generation task, which is initiated by the Testbed Controller. It then start the remaining tasks in the signal generation packages and serves as the communications interface between the RSS and the Testbed Controller.

4.1.1.1.1 Requirements

The following requirements have been allocated from the Testbed Specification [A-2]:

Number	Title
3.3.1.0.1	Multiple User Operation
3.3.1.0.2	Operation without DSP
3.3.2.0.1	Define Signal Generation Parameters
3.3.2.0.2	Generate Signal Data
3.3.2.0.5	Effects Modelled
3.3.2.0.6	Baseline Definition
3.3.2.0.7	Signal Processing Reconfigurability Definition
3.7.1.1.1	Interface Requirements
3.7.1.1.5	Signal Generation Parameters
5.1.0.0.1	Signal Generation Capability
5.1.1.1.1	Inter Burst Changes
5.1.1.1.2	Inter Dwell Changes
5.1.1.1.3	Sampling Rate
5.1.1.2.1	Pulse Modulation
5.1.1.2.2	Waveform type
5.1.1.10.2	Receiver Channels
5.3.0.0.2	Signal Generation (performance)

The following requirements have been allocated from the DSP/Testbed Interface Specification [A-1]:

Number	Title
3.2.1.2.2	Composition of 1 CPI
3.2.1.2.4	Data Element Definition
3.2.1.2.5	Number of channels

4.1.1.1.2 Design Description

The detailed design for the SSGM CSU is described in Appendix B7.1.
The PDL for the SSGM CSU is shown in Appendix E.

4.1.1.2 SGEN CSU

The SGEN CSU manages the overall signal generation activity and spawns the SRUN, SBIT, and SBMT CSUs.

4.1.1.2.1 Requirements

Not Applicable

4.1.1.2.2 Design Description

The detailed design for the SGEN CSU is described in Appendix B7.2.
The PDL for the SGEN CSU is shown in Appendix E.

4.1.1.3 SRUN CSU

The SRUN CSU is spawned by SGEN at the start of a signal generation activity, and performs all those functions required to be done only at the beginning of a run.

4.1.1.3.1 Requirements

Not Applicable

4.1.1.3.2 Design Description

The detailed design for the SRUN CSU is described in Appendix B7.3.
The PDL for the SRUN CSU is shown in Appendix E.

4.1.2 Burst Initialization CSC

The Burst Initialization CSC is fully decomposed into a single CSU, the Burst Initialization (SBIT) CSU.

The Burst Initialization CSC executes on the RSS workstation, and provides the functionality to set-up all of the data structures and variables to be used to calculate the signal return from a burst.

4.1.2.1 SBIT CSU

The SBIT CSU performs all those functions required to be performed once per coherent processing interval (CPI, or burst), and builds a burst database to be used by SBMT during execution.

4.1.2.1.1 Requirements

The following requirements have been allocated from the Testbed Specification [A-2]:

Number	Title
3.3.2.0.5	Effects Modelled (partial)
3.3.3.1.1	Data Logging
5.1.0.0.3	Effects Modelled
5.1.1.3.1	Earth Rotation
5.1.1.3.2	Spacecraft Simulation
5.1.1.4.1	Earth Clutter Map
5.1.1.4.2	Earth Clutter Models
5.1.1.4.3	Volumetric Clutter Map
5.1.1.4.4	Volumetric Clutter Models
5.1.1.7.1	Jammer Types
5.1.1.7.2	Jammer Distortion
5.1.1.8.1	Target Models
5.1.1.9.1	Noise Models
5.1.1.10.1	Quadratic Phase Modulation
5.1.1.10.3	Receiver Front-end
5.1.2.0.1	Sample Characteristics

Note: The partial allocation of requirements reflects the specification for general effects to be modelled, and the software design to split the functions into a setup (SBIT) and sample generation (SBMT) CSU.

4.1.2.1.2 Design Description

The detailed design for the SBIT CSU is described in Appendix B7.4.
The PDL for the SBIT CSU is shown in Appendix E.

4.1.3 Burst Modelling CSC

The Burst Modelling CSC is fully decomposed into a single CSU, the Burst Modelling CSU.

4.1.3.1 SBMT CSU

The SBMT CSU performs all of the calculations to determine the simulated radar signal data output for each burst.

4.1.3.1.1 Requirements

The following requirements have been allocated from the Testbed Specification [A-2]:

Number	Title
3.3.2.0.5	Effects Modelled(partial)
3.3.3.1.1	Data Logging
5.1.0.0.3	Effects Modelled (partial)
5.1.1.3.1	Earth rotation (partial)
5.1.1.3.2	Spacecraft simulation (partial)
5.1.1.4.1	Earth clutter map (partial)
5.1.1.4.2	Earth clutter models (partial)
5.1.1.4.3	Volumetric clutter map (partial)
5.1.1.4.4	Volumetric clutter models (partial)
5.1.1.7.1	Jammer Types (partial)
5.1.1.7.2	Jammer Distortion (partial)
5.1.1.8.1	Target Simulation
5.1.1.9.1	Noise Models
5.1.1.10.1	Quadratic Phase Modulation (partial)
5.1.1.10.3	Receiver Front-end
5.1.2.0.1	Sample Characteristics

Note: The partial allocation of requirements reflects the specification for general effects to be modelled, and the software design to split the functions into a setup (SBIT) and sample generation (SBMT) CSU.

4.1.3.1.2 Design Description

The detailed design for the SBMT CSU is described in Appendix B7.5.
The PDL for the SBMT CSU is shown in Appendix E.

4.1.4 Playback Manager CSC (PMT)

The Playback Manager CSC is fully decomposed into a single CSU, the Playback Manager CSU (SPMT).

4.1.4.1 Playback Manager CSU (SPMT)

The SPMT is the overall manager for the RSS playback function, and is initiated by the Testbed Controller, and acts as the interface between the RTSG and the Testbed Controller.

4.1.4.1.1 Requirements

The following requirements have been allocated from the Testbed Specification [A-2]:

Number	Title
3.3.2.0.3	Define Simulation Run Parameters
3.3.2.0.4	Playback Signal Data
5.1.0.0.2	Signal Playback Capability
3.3.1.0.1	Multiple User Operation

The following requirements have been allocated from the DSP/Testbed Interface Specification [A-1]:

Number	Title
3.2.1.1.4	Preconditions
3.2.1.2.1	Structure of Radar Signal Data

4.1.4.1.2 Design Description

The detailed design for the SPMT CSU is described in Appendix B7.7. The PDL for the SPMT CSU is shown in Appendix E.

4.1.5 DSP Interface CSC

The DSP Interface CSC is fully decomposed into 3 CSUs:

a) DSP Interface CSU (TDIT),
b) DSP Output CSU (TIOP), and
c) RPC Server CSU (TRSV).

4.1.5.1 DSP Interface CSU (TDIT)

The TDIT CSU controls the overall operation of the RTSG, by receiving and executing commands sent by TRSV. TDIT initiates the loading of run parameters and simulated radar signal data (SRSD), and then is capable of starting the playback activity, or aborting

4.1.5.1.1 Requirements

The following requirements have been allocated from the Testbed Specification [A-2]:

Number	Title

5.1.0.0.4	Signal Data Segment Duration
5.1.0.0.5	Data Playback Time
5.1.1.1.3	Sampling Rate
5.1.2.0.2	Data Segment Length
5.3.0.0.1	Signal Playback

The following requirements have been allocated from the DSP/Testbed Interface Specification [A-1]:

Number	Title
3.2.1.1.1	Concurrency
3.2.1.1.2	Sources and sinks
3.2.1.1.3	Application protocol

4.1.5.1.2 Design Description

The detailed design for the TDIT CSU is described in Appendix B7.9.
The PDL for the TDIT CSU is shown in Appendix E.

4.1.5.2 DSP Output CSU (TIOP)

The TIOP CSU is used to playback SRSD to the DSP, and operates under the control of the TDIT CSU.

4.1.5.2.1 Requirements

The following requirements have been allocated from the Testbed Specification [A-2]:

Number	Title
5.2.2.1.1	ESP/RSS Software Interface

4.1.5.2.2 Design Description

The detailed design for the TIOP CSU is described in Appendix B7.10.
The PDL for the TIOP CSU is shown in Appendix E.

4.1.5.3 RPC Server CSU (TRSV)

The TRSV CSU receives and passes on command received from the SPMT CSU in the form of RPC client calls, and forwards them to the TDIO CSU.

4.1.5.3.1 Requirements

Not Applicable.

4.1.5.3.2 Design Description

The detailed design for the TRSV CSU is described in Appendix B7.11.
The PDL for the TRSV CSU is shown in Appendix E.

SECTION FIVE

CONTROLLER WORKSTATION HWCI DETAILED DESIGN

5.1 HARDWARE OVERVIEW

The equipment block diagram for the Controller Workstation is shown in Figure 5-1. The major components of the Controller Workstation Hardware Configuration Item (HWCI) are as follows:

a) SPARCstation 10,
b) Keyboard,
c) Video Display,
d) Mouse,
e) Speakerbox and Microphone,
f) 150 MB Tape Drive,
g) 5 GB Tape Drive,
h) CD Drive,
i) Disk,
j) Printer,
k) ADS.

None of the major components require development.

A complete description of the major components is contained in the Testbed Preliminary Design document (R-3).

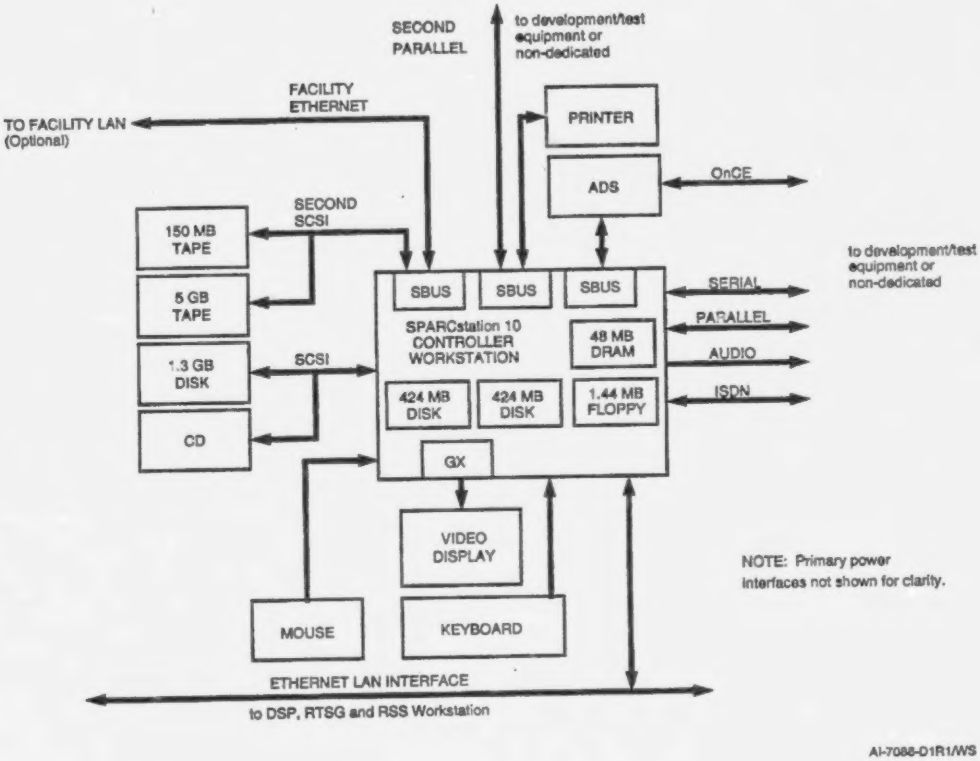

SECOND
PARALLEL
to development/test
equipment or
non-dedicated
FACILITY
ETHERNET
TO FACILITY LAN
(Optional)
PRINTER
ADS
OnCE
SECOND
SCSI
150 MB
TAPE
5 GB
TAPE
1.3 GB
DISK
CD
SCSI
SBUS
SBUS
SBUS
SPARCstation 10
CONTROLLER
WORKSTATION
48 MB
DRAM
424 MB
DISK
424 MB
DISK
1.44 MB
FLOPPY
GX
SERIAL
PARALLEL
AUDIO
ISDN
to development/test
equipment or
non-dedicated
VIDEO
DISPLAY
MOUSE
KEYBOARD
NOTE: Primary power
interfaces not shown for clarity.
ETHERNET LAN INTERFACE
to DSP, RTSG and RSS Workstation
AI-7086-D1R1/WS

Figure 5-1 Controller Workstation Equipment Diagram

5.2 HARDWARE DESIGN

None of the Controller Workstation major components require development, therefore this subsection will only describe differences or additions to the design as expressed in the Testbed Preliminary Design document (R-3).

5.2.1 SPARCstation 10

The SPARCstation 10 is a Sun model 30 which includes the following:

1. GX colour graphics,
2. Dynamic Random Access Memory (DRAM),
3. Internal SCSI hard disks,
4. Internal floppy disk drive,
5. Additional ethernet and Small Computer Systems Interface (SCSI),
6. Printer Card.

5.2.1.1 Requirements

5.2.1.1.1 Allocated Requirements.

The requirements allocated to the SPARCstation 10 are shown below. All requirements are traced to the Testbed Specification (Document A-2) unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification (Document A-1).

Number	Title
3.4.3.2.1	DSP Hardware Interface
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source
3.5.1.2.1	Graphics Controller
3.5.1.2.4	Non-volatile On-line Storage
4.1.2.1.1	Multitasking Environment
4.1.2.2.2	Interface Devices
4.1.2.3.2	Initialization
4.1.2.4.1	Floating-Point Computation
4.1.2.4.2	Integer Computation
4.2.1.2.1	Controller/HMI Hardware Interface
4.2.3.0.1	Controller/DSP Interface
4.3.2.0.2	Non-volatile On-line Storage
4.3.2.0.3	Graphics Controller
3.3.0.0.1 [I/F]	Allocation to Physical Links
3.3.0.0.2 [I/F]	Physical Connection
3.3.2.1.1 [I/F]	General
3.3.2.1.2 [I/F]	Physical Link Control
3.3.2.2.1 [I/F]	Cables and Connectors
3.3.2.3.1 [I/F]	Signal Levels

5.2.1.2 Interface Definition

The use of the SCSI and Second SCSI ports has been redefined as described in the following paragraphs.

5.2.1.2.1 SCSI (SCSI Port 1)

a) Interconnection. Daisy chain between devices using shielded multiple twisted-pair cable assemblies provided by Sun. Interconnection starts from a single connector on SPARCstation to the first SCSI device, then continuing through two connectors on each device until last device on daisy chain. Second connector on last device has a terminator installed.

b) SCSI Port 1 Devices. The external devices connected on SCSI Port 1 are as follows:

1. CD Drive,
2. Disk.

5.2.1.2.2 Second SCSI (SCSI Port 2)

a) Interconnection. Daisy chain between devices using shielded multiple twisted-pair cable assemblies provided by Sun. Interconnection starts from a single connector on SPARCstation to the first SCSI device, then continuing through two connectors on each device until last device on daisy chain. Second connector on last device has a terminator installed.

b) SCSI Port 2 Devices. The devices connected on SCSI Port 2 are as follows:

1. 150 MB Tape Drive,
2. 5 GB Tape Drive.

5.2.2 Keyboard

Sun type 5 Keyboard.

5.2.2.1 Requirements

5.2.2.1.1 **Allocated Requirements.** The requirements allocated to the Keyboard are shown below. All requirements are traced to the Testbed Specification (Document A-2) unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification (Document A-1).

Number	Title
3.4.2.0.2	Keyboard
4.1.2.3.2	Initialization
4.2.1.2.1	Controller/HMI Hardware Interface
4.3.2.0.1	HMI Performance

5.2.3 Video Display

Sun 19" Premium Color Monitor.

5.2.3.1 Requirements

5.2.3.1.1 Allocated Requirements.

The requirements allocated to the Video Display are shown below. All requirements are traced to the Testbed Specification (A-2) unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification (A-1).

Number	Title
3.4.2.0.1	Video Display
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source
4.2.1.2.1	Controller/HMI Hardware Interface
4.3.2.0.1	HMI Performance

5.2.4 Mouse

Sun Mouse.

5.2.4.1 Requirements

5.2.4.1.1 Allocated Requirements.

The requirements allocated to the Mouse are shown below. All requirements are traced to the Testbed Specification (A-2) unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification (A-1).

Number	Title
3.4.2.0.3	Pointing Device
4.2.1.2.1	Controller/HMI Hardware Interface
4.3.2.0.1	HMI Performance

5.2.5 Speakerbox and Microphone

Sun Speakerbox and Microphone.

5.2.6 150 MB Tape Drive

Sun 150-MByte 1/4" Tape Desktop Backup Pack.

5.2.6.1 Requirements

5.2.6.1.1 Allocated Requirements.

The requirements allocated to the 150 MB Tape Drive are shown below. All requirements are traced to the Testbed Specification (A-2) unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification (A–1).

Number	Title
3.4.2.0.5	Program Load/Store Device
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source
4.2.1.2.1	Controller/HMI Hardware Interface
4.3.2.0.1	HMI Performance

5.2.7 5 GB Tape Drive

Sun 5 GByte 8 mm Tape Drive Desktop Storage Module.

5.2.7.1 Requirements

5.2.7.1.1 Allocated Requirements.

The requirements allocated to the 5 GB Tape Drive are shown below. All requirements are traced to the Testbed Specification (A-2) unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification (A-1).

Number	Title
3.4.2.0.6	Archive Device
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source
4.2.1.2.1	Controller/HMI Hardware Interface
4.3.2.0.1	HMI Performance

5.2.8 CD Drive

Sun 644 MBYTE CD-ROM Desktop SunCD Pack.

5.2.8.1 Requirements

5.2.8.1.1 Allocated Requirements.

The requirements allocated to the CD Drive are shown below. All requirements are traced to the Testbed Specification (A-2) unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification (A-1).

Number	Title
3.4.2.0.5	Program Load/Store Device
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source
4.2.1.2.1	Controller/HMI Hardware Interface
4.3.2.0.1	HMI Performance

5.2.9 Disk

Sun 1.3 GBYTE SCSI Disk Desktop Storage Module.

5.2.9.1 Requirements

5.2.9.1.1 Allocated Requirements.

The requirements allocated to the Disk are shown below. All requirements are traced to the Testbed Specification (A-2) unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification (A-1).

Number	Title
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source
3.5.1.2.4	Non-volatile On-line Storage
4.3.2.0.2	Non-volatile On-line Storage

5.2.10 Printer

Sun SPARCprinter.

5.2.10.1 Requirements

5.2.10.1.1 **Allocated Requirements**. The requirements allocated to the Printer are shown below. All requirements are traced to the Testbed Specification (A-2) unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification (A-1).

Number	Title
3.4.2.0.4	Printer
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source
4.2.1.2.1	Controller/HMI Hardware Interface
4.3.2.0.1	HMI Performance

5.2.11 ADS

Motorola 96002 Application Development System.

5.2.12 Interchangeability (All Major Components)

The Workstation and peripherals are all standard commercial items. These items are interchangeable at the equipment unit level only with a like type and like configured item. There are no common user interchangeable items between equipment units of different types.

5.3 TRANSPORTABILITY, PACKAGING, HANDLING

5.3.1 Transportability

The Controller Workstation equipment is designed for permanent installation at the customer's site and is not designed for frequent transportation between sites. Infrequent movement within the interior of the same building on the same site is permissible without repackaging.

5.3.2 Packaging

The Controller Workstation equipment should be packaged for shipping to the customer in the same containers in which it was received from the vendor for maximum protection (corrugated cardboard box with polystyrene foam cushioning). Equivalent or better packaging may be used if the original containers are not available or are not reusable.

5.3.3 Handling

Some Controller Workstation equipment may require two people to lift.

Any commercial mode of transport may be used so long as it provides protection from moisture, shock, extreme pressure or temperature variations, and strong electromagnetic or electrostatic fields.

5.4 SUPPORT EQUIPMENT REQUIREMENTS

These items are in addition to any deliverable equipment which may be used in a support role.

5.4.1 Development and Test

The following items are required for development and test of the Controller Workstation:

a) Corporate computing facilities (printers, terminals, network, computers);
b) Systems room.

5.4.2 Installation

The following items are required for installation of the Controller Workstation:

a) Standard tool-kit (MacDonald Dettwiler issued).

5.4.3 Maintenance

The following items are required for maintenance of the Controller Workstation:

a) Clean, dry , lint-free cloths for cleaning cabinet exteriors and CDs;

b) Cotton swabs and tape head cleaning fluid for cleaning 150 MB Tape Drive heads;

c) Cleaning kit for 5 GB Tape Drive;

d) Toner cartridges for printer (1 replacement provided);

e) Drum cartridges for printer (1 replacement provided).

SECTION 6

RSS WORKSTATION HWCI DETAILED DESIGN

6.1 HARDWARE OVERVIEW

The equipment block diagram for the RSS Workstation is shown in Figure 6-1. The major components of the RSS Workstation Hardware Configuration Item (HWCI) are as follows:

a) SPARCstation 10,
b) Keyboard,
c) Video Display,
d) Mouse,
e) Speakerbox and Microphone,
f) 5 GB Tape Drive,
g) CD Drive,
h) two 1.3 GByte Disks,
i) ADS, and
j) Vector Co-processor.

None of the major components require development.

A complete design description of the major components is contained in the Testbed Preliminary Design Document [R-3].

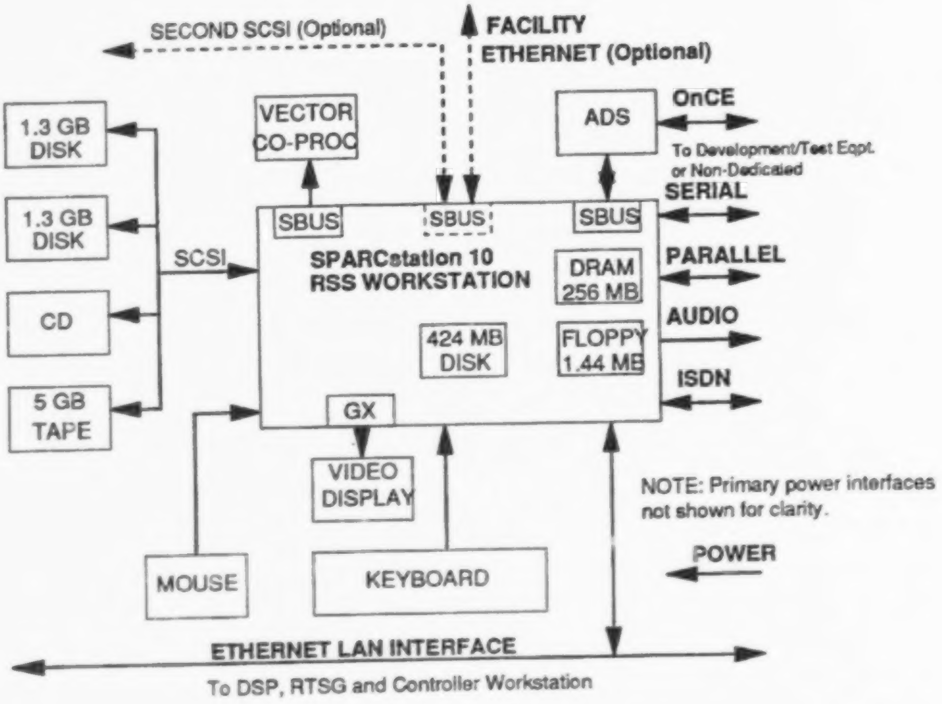

SECOND SCSI (Optional)
FACILITY
ETHERNET (Optional)
1.3 GB DISK
1.3 GB DISK
CD
5 GB TAPE
SCSI
VECTOR CO-PROC
SBUS
SBUS
SBUS
ADS
OnCE
To Development/Test Eqpt. or Non-Dedicated
SERIAL
PARALLEL
AUDIO
ISDN
SPARCstation 10
RSS WORKSTATION
DRAM 256 MB
424 MB DISK
FLOPPY 1.44 MB
GX
VIDEO DISPLAY
MOUSE
KEYBOARD
NOTE: Primary power interfaces not shown for clarity.
POWER
ETHERNET LAN INTERFACE
To DSP, RTSG and Controller Workstation

Figure 6-1 RSS WORKSTATION HWCI BLOCK DIAGRAM

6.2 HARDWARE DESIGN

None of the RSS Workstation major components require development, therefore this subsection will only describe differences or additions to the design as expressed in the Testbed Preliminary Design document [R-3].

6.2.1 SPARCstation 10

The SPARCstation 10 is a Sun model 41 which includes the following:

a) GX colour graphics,
b) 256 MB DRAM,
c) Internal 424 MB SCSI hard disk, and
d) Internal floppy disk drive,

6.2.1.1 Requirements

6.2.1.1.1 **Allocated Requirements.** The requirements allocated to the SPARCstation 10 are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.4.3.2.1	DSP Hardware Interface
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source
3.5.1.2.1	Graphics Controller
3.5.1.2.4	Non-volatile on-line storage
5.2.1.2.1	Controller/RSS Hardware Interface
3.3.0.0.1 [I/F]	Allocation to physical links
3.3.0.0.2 [I/F]	Physical connection
3.3.2.1.1 [I/F]	General
3.3.2.2.1 [I/F]	Cables and connectors
3.3.2.3.1 [I/F]	Signal levels

6.2.2 Keyboard

Sun type 5 keyboard.

6.2.2.1 Requirements

6.2.2.1.1 **Allocated Requirements.** The requirements allocated to the keyboard are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.4.2.0.2	Keyboard

6.2.3 Video Display

Sun 19" Premium Color Monitor

6.2.3.1 Requirements

6.2.3.1.1 **Allocated Requirements.** The requirements allocated to the video display are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.4.2.0.1	Video Display
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source

6.2.4 Mouse

Sun Mouse.

6.2.4.1 Requirements

6.2.4.1.1 **Allocated Requirements.** The requirements allocated to the mouse are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.4.2.0.3	Pointing Device

6.2.5 Speakerbox and Microphone

Sun Speakerbox and Microphone.

6.2.6 5 GB Tape Drive

Sun 5 GByte 8mm Tape Drive Desktop Storage Module.

6.2.6.1 Requirements

6.2.6.1.1 **Allocated Requirements.** The requirements allocated to the tape drive are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.4.2.0.6	Archive Device
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source

6.2.7 CD Drive

Sun 644 MByte CD-ROM Desktop SunCD Pack.

6.2.7.1 Requirements

6.2.7.1.1 **Allocated Requirements.** The requirements allocated to the CD drive are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.4.2.0.6	Program Load/Store Device
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source

6.2.8 Disk

Sun 1.3 GByte SCSI Disk Desktop Storage Module (Quantity 2).

6.2.8.1 Requirements

6.2.8.1.1 **Allocated Requirements.** The requirements allocated to the disks are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source
3.5.1.2.4	Non volatile on line storage

6.2.9 ADS

Motorola 96002 Application Development System.

6.2.10 Vector Co-processor

CSPI Supercard-S RISC Processor

6.2.10.1 Requirements

6.2.10.1.1 **Allocated Requirements.** The requirements allocated to the co-processor are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
5.3.0.0.2	Performance - Signal Generation

6.2.11 Interchangeability (All Major Components)

The Workstation and peripherals are all standard commercial items. These items are interchangeable at the equipment unit level only with a like type and like configured item. There are no common user interchangeable items between equipment units of different types.

6.3 TRANSPORTABILITY, PACKAGING, HANDLING

6.3.1 Transportability

The RSS Workstation equipment is designed for permanent installation at the customer site and is not designed for frequent transportation between sites. Infrequent movement within the interior of the same building on the same site is permissible without repackaging.

6.3.2 Packaging

The RSS Workstation equipment should be packaged for shipping to the customer in the same containers in which it was received from the vendor for maximum protection (corrugated cardboard box with polystyrene foam cushioning). Equivalent or better packaging may be is used if the original containers are not available or are not reuseable.

6.3.3 Handling

Some RSS Workstation equipment may require two people to lift.

All commercial mode of transport may be used so long as it provides protection from moisture, shock, extreme pressure or temperature variations, and strong electromagnetic or electrostation fields.

6.4 SUPPORT EQUIPMENT REQUIREMENTS

These items are in addition to any deliverable equipment which may be used in a support role.

6.4.1 Development and Test

The following items are required for development and test of the RSS Workstation:

a) Corporate computing facilities (printers, terminals, network, computers), and
b) Systems room.

6.4.2 Installation

The following items are required for installation of the RSS Workstation:

a) Standard took-kit (MacDonald Dettwiler issued).

6.4.3 Maintenance

The following items are required for maintenance of the RSS Workstation:

a) Clean, dry, lint-free cloths for cleaning cabinet exteriors and CDs,
b) Cleaning kit for 5 GB Tape Drive,

SECTION 7

RTSG HWCI DETAILED DESIGN

7.1 HARDWARE OVERVIEW

The equipment block diagram and chassis slot allocation diagram for the Real Time Signal Generator (RTSG) is shown in Figures 7-1 and 7-2. The major components of the RTSG Hardware Configuration Item (HWCI) are as follows:

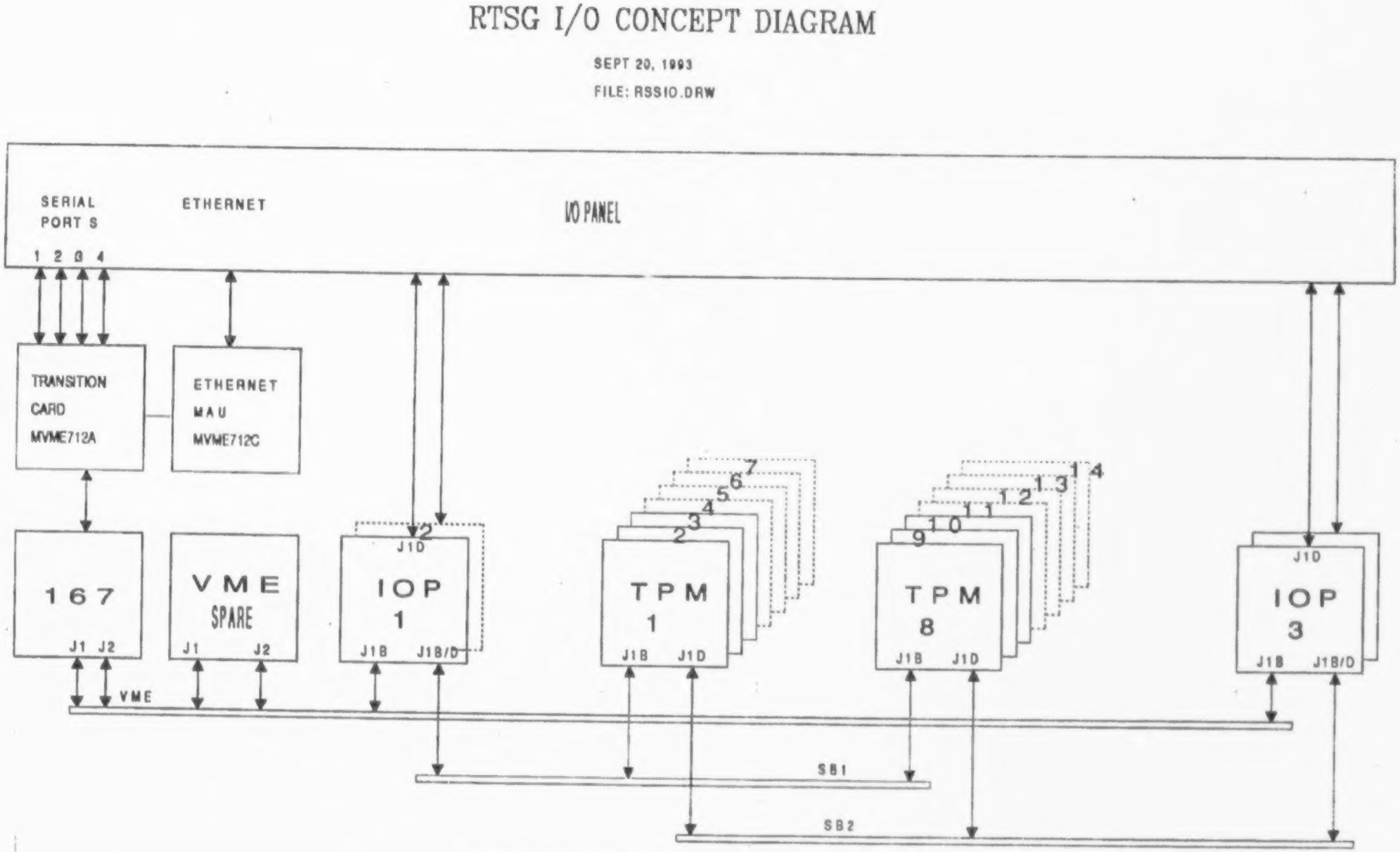
a) Executive Controller (EC),
b) Input-Output Processors (IOP),
c) Two-Port Memories (TPM),
d) Card Cage,
e) Backplane,
f) Power Supply,
g) Chassis,
h) Cooling Fans and Filter,
i) IOP Adapter.

The only purely developmental items are the Backplane and IOP Adapter.

The Card Cage, Power Supply, Chassis, and Cooling Fans and Filter are all part of a modified Commercial Off-The-Shelf (COTS) product supplied by Mupac.

The EC and IOPs are fully COTS except for the addition of Erasable Programmable Read Only Memories (EPROM) containing developed or third-party firmware, and keying parts in the case of the GSPs and IOPs.

A complete description of the non-developmental portions of the major components is contained in the Testbed Preliminary Design [R-3].



RTSG I/O CONCEPT DIAGRAM

SEPT 20, 1993
FILE: RSSIO.DRW

SERIAL
PORT S
1 2 3 4
ETHERNET
I/O PANEL
TRANSITION
CARD
MVME712A
ETHERNET
MAU
MVME712C
167
J1 J2
VME
SPARE
J1
J2
IOP
1
2
J1D
J1B
J1B/D
TPM
1
2
3
4
5
6
7
J1B
J1D
TPM
8
9
10
11
12
13
14
J1B
J1D
IOP
3
J1D
J1B
J1B/D
VME
SB1
SB2

Figure 7-1 RTSG EQUIPMENT DIAGRAM

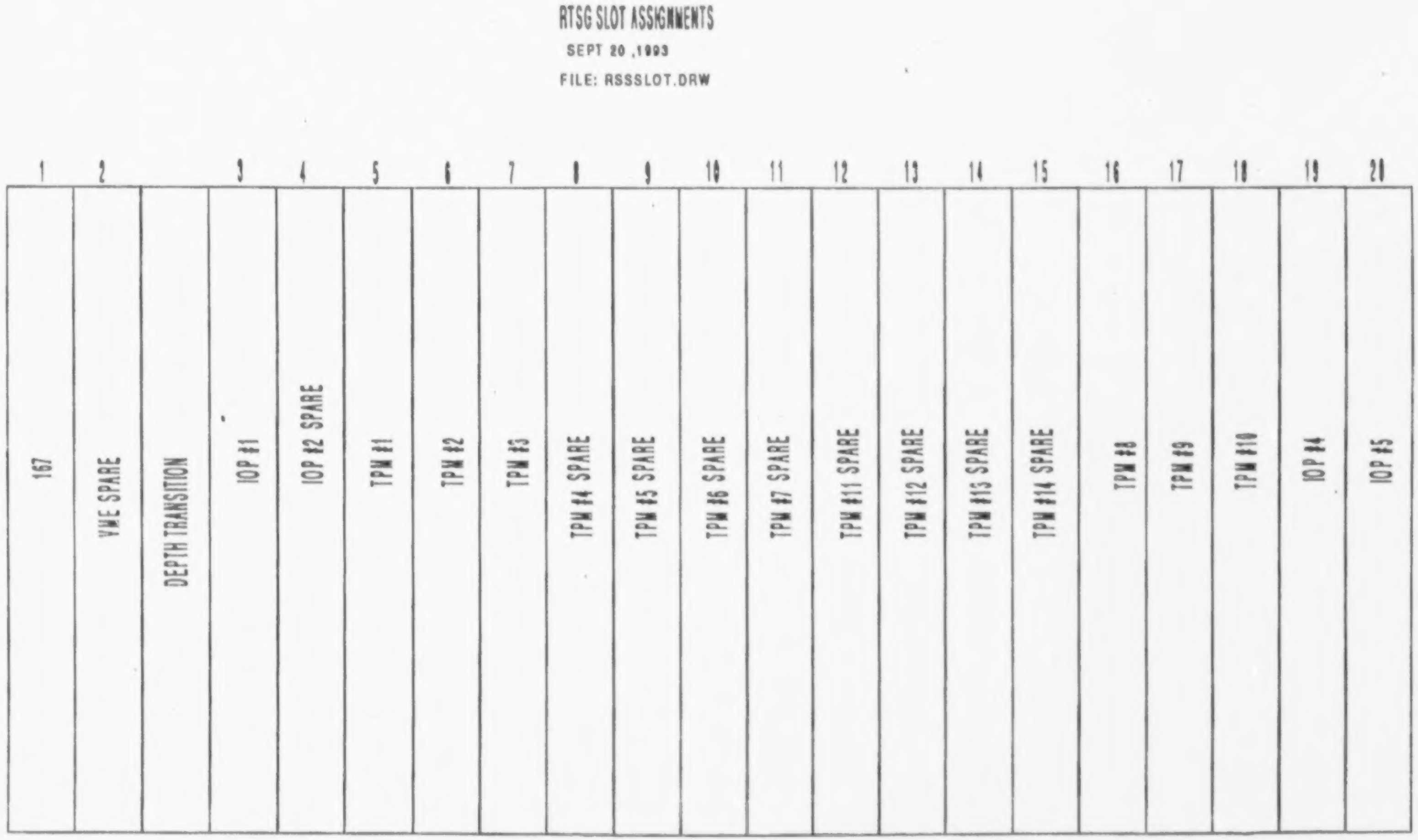

RTSG SLOT ASSIGNMENTS
SEPT 20 ,1993
FILE: RSSSLOT.DRW
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
167
VME SPARE
DEPTH TRANSITION
IOP #1
IOP #2 SPARE
TPM #1
TPM #2
TPM #3
TPM #4 SPARE
TPM #5 SPARE
TPM #6 SPARE
TPM #7 SPARE
TPM #11 SPARE
TPM #12 SPARE
TPM #13 SPARE
TPM #14 SPARE
TPM #8
TPM #9
TPM #10
IOP #4
IOP #5

Figure 7-2 RTSG CHASSIS SLOT ALLOCATION DIAGRAM

7.2 HARDWARE DESIGN

This section must be read in conjunction with the Testbed Preliminary Design [R-3] for a complete description of the design.

This section will describe the design in the following manner:

a) Developmental Items: Complete description of the design decomposition plus the description of any changes or additions to the design as expressed in the Testbed Preliminary Design [R-3].

b) Modified COTS: Complete description of the modifications plus the description of any changes or additions to the design as expressed in the Testbed Preliminary Design [R-3].

c) COTS: Description of any changes or additions to the design as expressed in the Testbed Preliminary Design [R-3].

The RTSG Hardware Design Documentation Library is contained in Appendix F. The items included are the Parts List, Schematics and Assembly Drawings. The RTSG Backplane Hardware Design Documentation Library comprising the Build Standard List, Parts List, Schematics, Fabrication Drawings and Assembly Drawings, is contained in Appendix H.

7.2.1 Executive Controller

The EC is composed of a Motorola MVME167 Single-Board-Computer, MVME712A serial/parallel transition module, MVME712C ethernet/SCSI transition module, and MVME147P2 adapter board. The MVME167 is installed in the backplane/card cage. The MVME712 modules are installed in the rear panel of the Chassis. The MVME147P2 adapter board is installed into the rear of the backplane.

7.2.1.1 Requirements

7.2.1.1.1 **Allocated Requirements.** The requirements allocated to the EC are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.3.0.0.1 [I/F]	Allocation to Physical Links
3.3.0.0.2 [I/F]	Physical Connection
3.3.2.1.1 [I/F]	General
3.3.2.1.2 [I/F]	Physical Link Control
3.3.2.2.1 [I/F]	Cables and Connectors
3.3.2.3.1 [I/F]	Signal Levels.
5.2.1.2.1	Controller-RSS Hardware Interface

7.2.1.2 Interface Definition

7.2.1.2.1 **SCSI.** The EC provides a SCSI interface compliant with SCSI Small Computer Systems Interface X3T9.2/82-2 [R-5]. The pinout of this interface is shown in the MVME712A User's Manual [R-6]. The rear panel connector for this interface and the internal cable to the MVME147P2 adapter is not provided.

7.2.1.2.2 **Mechanical.**

a) Dimensions. The mechanical characteristics of the MVME712 transition cards are shown in the MVME712A User's Manual [R-6].

b) Assembly. Assembly and interconnection of the transition cards is shown in the MVME712A User's Manual [R-6] and the RTSG Chassis Schematics (SDG442051) and Assembly Drawings (G442051) in Appendix F.

7.2.1.3 Performance Characteristics

7.2.1.3.1 **Processor.** The MVME167 contains a single MC68040 processor with the following performance:

a) Clock Speed (MHz): 33,
b) Instruction Rate (MIPS): 34,
c) Floating-Point Operation Rate (MFLOPS): 4.6.

7.2.1.3.2 **DRAM Support.** The MVME167 supports up to two of the following DRAM modules supplied by the vendor:

a) 4 MB;
b) 8 MB (as delivered);
c) 16 MB;
d) 32 MB.

The total memory supplied with the MVME167 is 8 MB.

7.2.1.3.3 **EPROM Support.** The MVME167 supports the following EPROM devices as supplied by the vendor:

a) 1 Mb (64k x 16), 27C102 or equivalent type in Plastic Leaded Chip Carrier (PLCC) or Ceramic Leaded Chip Carrier (CLCC) package;
b) 2 Mb (128k x 16), 27C202 or equivalent type in PLCC or CLCC package.

7.2.1.3.4 **Power Consumption.** The maximum power consumption of the EC (includes MVME167, MVME712A, and MVME712C) is as follows:

a) 5V at 4.8 A;
b) +12V at 1.0 A;
c) -12V at 0.1 A.

7.2.1.4 Physical Characteristics

7.2.1.4.1 **Size.** The approximate mechanical dimensions of the EC are as follows:

a) MVME167: 6U x 160 mm x 1 slot (4HP), standard VME card size;
b) MVME712A: 6U x 80 mm x 1 slot (4HP);
c) MVME712C: 6U x 80 mm x 2 slot (8HP);
d) MVME147P2: 87 mm x 51 mm x 15 mm.

7.2.1.4.2 **Weight.** The estimated weight of the EC is as follows:

a) MVME167: 1 Kg;
b) MVME712A: 0.5 Kg;
c) MVM712C: 0.5 Kg;
d) MVME147P2: included in MVME712A.

7.2.1.5 Expandability

The EC can be expanded in the following ways:

a) EPROM: the MVME167 supports up to four EPROM devices as specified in 7.2.1.3.3. Larger EPROM devices, up to 8 Mb, can be supported if a minor board modification is performed (see MVME167 Single Board Computer Support Information [R-7]).

b) SCSI Interface: the external connector and internal cabling can be added to the MVME712C and MVME147P2 to supply an external connection to the MVME167 SCSI interface (see MVME167 Single Board Computer Support Information [R-7] and MVME712 Transition Module and MVME147P2 Adapter Board User's Manual [R-6]).

c) DRAM: can be expanded up to 64 Mb using modules described in 7.2.1.3.2. Note that a hardware upgrade is required to support two modules.

7.2.1.6 Thermal Design

The EC dissipates approximately 37 W.

7.2.1.7 Reliability

Motorola has supplied the reliability factor in terms of availability over time for another similar program. The MVME167 is rated at 90% availability for 83522 hours of use.

7.2.1.8 Interchangeability

The MVME167 is a standard commercial product. It can be replaced by the user with an identical product identically configured (see RTSG Chassis Assembly Drawing G442051 in Appendix F for configuration information).

7.2.1.9 Maintainability

The MVME167 is shipped with a monitor program in firmware which provides basic board level testing (see MVME167BUG Debugging Package User's Manual [R-13]).

7.2.2 Input-Output Processor

The IOP is a Spectrum VASP product. It contains a single DSP96002 processor, VMEbus Interface, Signal Bus Interface and Synchronous Parallel Bus Interface.

7.2.2.1 Requirements

7.2.2.1.1 Allocated Requirements

The requirements allocated to the IOP are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-2].

Number	Title
3.4.3.2.1	DSP Hardware Interface
3.5.1.2.3	Real Time Output

7.2.2.2 Interface Definition

7.2.2.2.1 **Synchronous Parallel Bus Interface.** The modification made to the SPBI to add the hardware flow control function is described in the DSP/Testbed Interface Detailed Design document [R-3].

7.2.2.2.2 **Connector Keying.** The IOP has two coding parts installed on the connector J1 (see Appendix I of DSP Detailed Design Document [R-2], IOP Assembly Drawing 28-0553). This keying permits installation in IOP compatible slots only.

7.2.2.3 Performance Characteristics

7.2.2.3.1 **Processor.** The IOP contains one Motorola 96002 processor with a peak floating point operation rate of 40 MFLOPS.

7.2.2.3.2 **SRAM.** The IOP has 128 kbyte of SRAM.

7.2.2.3.3 **EPROM Support.** The IOP supports the following EPROM device:

a) 1 Mb (128k x 8) 27C010 or equivalent type in ceramic DIP package.

7.2.2.3.4 **Interfaces.** The IOP supports the following interfaces and data rates.

a) VMEbus: 10.7 Mbytes/sec.
b) Signal Bus: 35 Mbytes/sec write and read.
c) Synchronous Parallel Bus: 26 Mbytes/sec.

7.2.2.3.5 **Environment.** The IOP has the following environmental performance characteristics

a) Operating Temperature: (°C): 0 - 40; (see 7.2.7.5)
b) Storage Temperature (°C): -40 - 85;
c) Operating Humidity (%RH): 5 - 90 non-condensing;
d) Storage Humidity (%RH): 5 - 90 non-condensing;
e) Maximum operating altitude (m): 3000;
f) Maximum storage altitude (m): 15,000.

7.2.2.3.6 **Power Consumption.** The maximum power consumption of the IOP is as follows:

a) 5V at 5.2 A.

7.2.2.4 Physical Characteristics

7.2.2.4.1 **Size.** The approximate mechanical dimensions of the IOP are 6U x 220 mm x 1 slot (.8 inch), standard VASP card size.

7.2.2.4.2 **Weight.** The estimated weight of the IOP is 1 Kg.

7.2.2.5 Thermal Design

The IOP dissipates approximately 26 W. (also see 7.2.7.5)

7.2.2.6 Reliability

The reliability of the IOP has been calculated previously for another similar program. The MTBF predicted for the IOP is greater than 45,000 hours.

7.2.2.7 Interchangeability

The IOP is a standard commercial product. It can be replaced by the user with an identical product identically configured (see RTSG Chassis Assembly Drawing G442051 in Appendix F for configuration information).

7.2.2.8 Maintainability

The boot software on the IOP contains some basic hardware tests which are automatically executed each time the IOP is reset (see the VASP Technical Manual [R-10]). These tests include; memory test, FPGA load complete and checksum test, lamp (LED) test, and load from EPROM checksum test. Additional testing can be performed using the Application Development System (ADS) (see Application Development System Reference Manual [R-11]). Various test points are provided for low level testing (see the VASP Technical Manual [R-10]). Testing using the ADS and test points is intended to be performed by trained technical personnel only and is not intended for user level diagnostic testing.

7.2.3 Two Port Memory

The TPM is a Spectrum VASP product. It contains from 8 to 32 MB of DRAM which is dual ported to two Signal Bus Interfaces.

7.2.3.1 Requirements

7.2.3.1.1 **Allocated Requirements.** The requirements allocated to the TPM are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.5.1.2.5	High Speed Memory Capacity

7.2.3.2 Interface Definition

7.2.3.2.1 **Connector Keying.** The TPM has two coding parts installed on the connector J1 (see Appendix I of the DSP Detailed Design Document [R-2], TPM Assembly Drawing 28-0555). This keying permits installation in TPM compatible slots only.

7.2.3.3 Performance Characteristics

7.2.3.3.1 **Memory.** The TPM is configured with up to 32 Mbytes of DRAM in 8 Mbyte increments. The six RTSG TPMs are each configured for 32 MBytes.

7.2.3.3.2 **Interfaces.** The TPM connects two signal buses having a burst transfer rate of 86 Mbytes/sec.

7.2.3.3.3 **Environment.** The TPM has the following environmental performance characteristics

a) Operating Temperature: (°C): 0 - 40; (see 7.2.7.5)
b) Storage Temperature (°C): -40 - 85;
c) Operating Humidity (%RH): 5 - 90 non-condensing;
d) Storage Humidity (%RH): 5 - 90 non-condensing;
e) Maximum operating altitude (m): 3000;
f) Maximum storage altitude (m): 15,000.

7.2.3.3.4 **Power Consumption.** The maximum power consumption of the TPM is as follows:

a) 5V at 4.3 A.

7.2.3.4 Physical Characteristics

7.2.3.4.1 **Size.** The approximate mechanical dimensions of the TPM are 6U x 220 mm x 1 slot (.8 inch), standard VASP card size.

7.2.3.4.2 **Weight.** The estimated weight of the TPM is 1 Kg.

7.2.3.5 Thermal Design

The TPM dissipates approximately 22 W. (also see 7.2.7.5)

7.2.3.6 Reliability

The reliability of the TPM has been calculated previously for another similar program. The MTBF predicted for the TPM is greater than 45,000 hours.

7.2.3.7 Interchangeability

The TPM is a standard commercial product. It can be replaced by the user with an identical product identically configured (see RTSG Chassis Assembly Drawing G442051 in Appendix F for configuration information). The memory devices on the TPM are socketed so that banks of memory can be added or removed easily.

7.2.3.8 Maintainability

The TPM has a test interface for low level testing (see the VASP Technical Manual [R-10]). This test interface is intended for use by trained technical personnel and is not intended for user level diagnostic testing.

7.2.4 Card Cage

The Card Cage is a modified COTS product provided by Mupac along with the Chassis, Cooling Fans and Filter, and Power Supply.

7.2.4.1 Interface Definition

7.2.4.1.1 **Mechanical.** The mechanical layout and dimensions (inch) of the Card Cage are shown in Figure 7-3 of the DSP Detailed Design Document. This figure is derived from the Chassis specification supplied to Mupac.

7.2.4.2 Performance Characteristics

7.2.4.2.1 **Card Slots.** The Card Cage accommodates the following cards:

a) 2 VME (6U x 160 mm),
b) 18 VASP (6U x 220 mm).

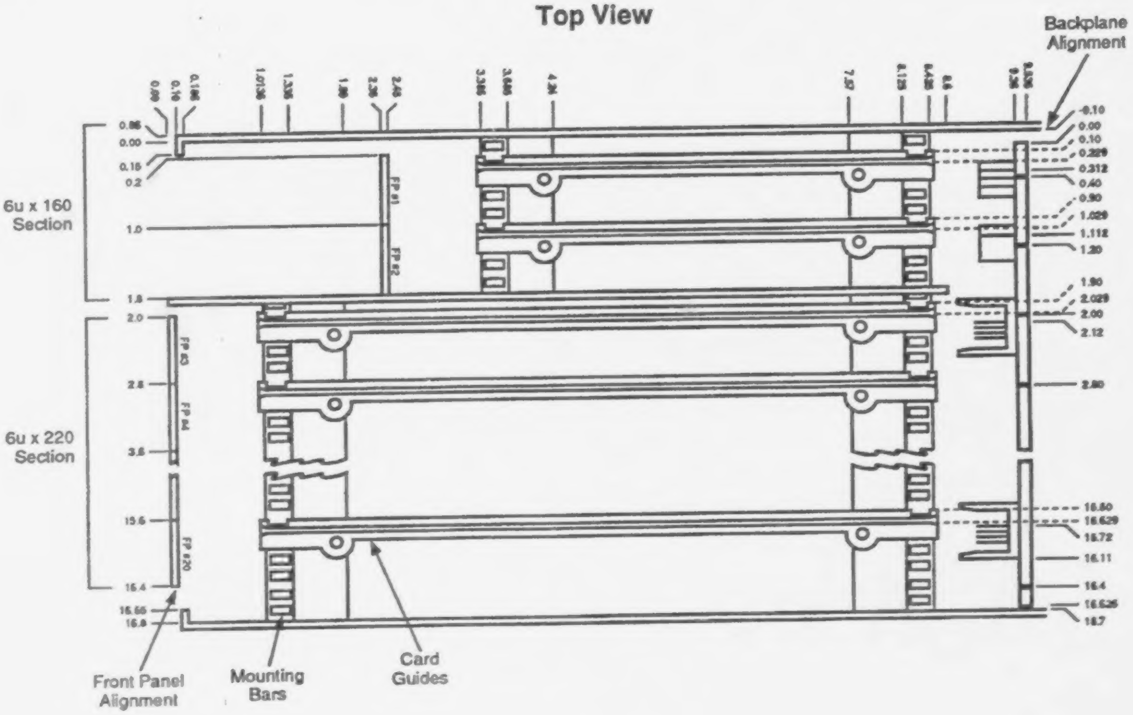
7.2.4.3 Physical Characteristics

7.2.4.3.1 **Size.** The approximate size of the Card Cage is 42.7 cm wide x 26.5 cm high x 23.8 cm deep.

7.2.4.3.2 **Weight.** The estimated weight of the Card Cage is 2 Kg.

7.2.4.4 Electromagnetic Compatibility

The EMC of the Card Cage is enhanced by the grounding of the circuit board front panels to the card cage frame. Unused slots have filler panels installed.


Top View
Backplane Alignment
0.09
0.16
0.186
1.0136
1.336
1.96
2.36
2.46
3.386
3.686
4.36
7.57
8.125
8.426
8.6
9.26
9.526
-0.10
0.00
0.10
0.229
0.312
0.40
0.90
1.029
1.112
1.20
1.90
2.029
2.00
2.12
2.80
16.50
16.629
16.72
16.11
16.4
16.626
16.7
0.06
0.00
0.15
0.2
1.0
1.8
2.0
2.8
3.6
15.6
16.4
16.56
16.8
6u x 160 Section
6u x 220 Section
FP #1
FP #2
FP #3
FP #4
FP #20
Front Panel Alignment
Mounting Bars
Card Guides

Dimensions in Inches

AI-7007-D6R2/CD

Figure 7-3 CARD CAGE MECHANICAL LAYOUT

7.2.5 Backplane

The backplane is a developmental item. It is manufactured by Symtron Corporation.

7.2.5.1 Requirements

7.2.5.1.1 **Allocated Requirements.** The requirements allocated to the Backplane are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.3.0.0.1 [I/F]	General
3.3.2.1.1 [I/F]	Physical Link Control
3.3.2.1.2 [I/F]	Signal Levels
3.3.2.3.1 [I/F]	RSS-ESP Hardware Interface

7.2.5.2 Interface Definition

7.2.5.2.1 Mechanical

a) Fabrication. The spacings (inch) for the backplane mounting holes and connector positions are shown in Figure 7-3 of the DSP Detailed Design Document. The complete mechanical details of the Backplane are shown on the RTSG Motherboard Fabrication Drawing 28-0561 (see Appendix H).

b) Assembly. The placement and assembly of parts onto the backplane is detailed on the RTSG Motherboard Assembly Drawing 28-0495 (see Appendix H). The physical interconnection of the other subassemblies with the Backplane are shown in the RTSG Chassis Schematic SDG442051 and Assembly Drawings G442051 (see Appendix F).

7.2.5.2.2 **VMEbus.** A VMEbus Timeout function is implemented on the Backplane according to the VMEbus Specification [R-8].

7.2.5.2.3 **Reset.** The Backplane is electrically interfaced to the Reset Switch. The wiring details are shown on the RTSG Chassis Subassembly Schematic Drawing (SDG442051) (Appendix F). The reset switch is debounced by circuitry on the Backplane.

7.2.5.2.4 **DC Power.** The Backplane is electrically interfaced to the Power Supply. The wiring details are shown on the RTSG Chassis Subassembly Schematic Drawing SDG442051 (Appendix F).

7.2.5.3 Performance Characteristics

7.2.5.3.1 **VMEbus Timeout.** The timeout period of the VME Bus Timer is 18.5 ms. This exceeds the minimum the timeout period of 2 ms as specified in the VASP Technical Manual [R-10].

7.2.5.3.2 **Reset Switch Debounce.** The debounce period of the Reset Switch Debouncer is 126 ms.which is more than double the rule-of-thumb worst case of 50 ms.

7.2.5.3.3 **Backplane Loaded Impedance.** The loaded impedance with a fully loaded backplane is as follows:

a) VMEbus: 23.8 ohms;
b) Signal Bus: 23.6 ohms;
c) Signal Bus Clocks: 46.4 ohms.

The maximum loaded impedance with no boards installed is as follows:

d) VMEbus: 51.0 ohms;
e) Signal Bus: 48.6 ohms;
f) Signal Bus Clocks: 58.7 ohms.

Note the loaded impedance varies inversely proportional to the square root of the ratio of added capacitance per unit length versus transmission line capacitance per unit length.

The backplane unloaded impedance for all signal lines is 65 ohms nominal. The following geometries are used to achieve this impedance:

1. Trace width: 5 mils;
2. Trace thickness: 1.25 mils (1 oz copper after microetching);
3. Trace separation: 33 mils;
4. Dielectric thickness: 12 mils;
5. Dielectric constant: 4.0 (effective for FR4 fibreglass reinforced epoxy laminate).

7.2.5.3.4 **Power Consumption.** The maximum power consumption of the backplane is as follows:

a) 5V: 1.8 A,
b) 2V: 15.0 A

7.2.5.4 Physical Characteristics

7.2.5.4.1 **Size.** The approximate size of the Backplane is 42.2 cm wide x 26.2 high x 0.65 cm thick (board only, no components).

7.2.5.4.2 **Weight.** The approximate weight of the Backplane is 3 Kg.

7.2.5.5 Thermal Design

The Backplane dissipates 39 W.

7.2.5.6 Power Distribution and Grounding

The Backplane has the following power distribution network (see Appendix F, RTSG Chassis Subassembly Schematic Drawing SDG442051):

a) +5V: 3 layers, 13 power entry terminals (4-40 press-nuts);
b) +2V: 2 layers, 8 power entry terminals (4-40 press-nuts);
c) +12V: heavy traces, 1 power entry terminals (2-56 press-nuts);
d) -12V: heavy traces, 1 power entry terminals (2-56 press-nuts);
e) GND: 5 layers, 16 power entry terminals (4-40 press-nuts), 2 power entry terminals (2-56 press-nuts);
f) +5V sense: 1 sense terminal (2-56 press-nut);
g) +2V sense: 1 sense terminal (2-56 press-nut).

The sense terminals are located so that they are not immediately adjacent to a power entry terminal for the voltage they are sensing.

7.2.5.7 Interchangeability

The number of slots which are equipped to handle the IOP Adapter is limited to a maximum of six.

7.2.5.8 Signal Bus Clock Generator Circuit Design

7.2.5.8.1 **Context Description.** The Signal Bus Clock Generator provides a clock and clock qualifier signal to each VASP board slot.

7.2.5.8.2 **Functional Description.** An independent synchronized clock/clock qualifier pair is generated by the Signal Bus Clock Generator for each VASP slot in the Backplane. The VASP boards use the clock/clock qualifier to synchronize their Signal Bus transactions.

7.2.5.8.3 **Physical Description.** The Signal Bus Clock Generator provides the following TTL clock/clock qualifier pairs on the Backplane:

a) CLK<0:17>: 25 MHz clock, 1 instance for each VASP slot, trace lengths are matched to minimize clock skew between slots;

b) CLKQ<0:17>: 12.5 MHz clock qualifier, 1 instance for each VASP slot, trace lengths are matched to minimize clock skew between slots.

7.2.5.8.4 **Electrical Interface.** The timing of the clock signals is shown in Figure 7-4. The electrical characteristics of the clock signals are defined in the VASP Technical Manual [R-10].

7.2.5.8.5 **Circuit Operation.** (RTSG Motherboard Subassembly Schematic Drawing 28-0494, Appendix H): A crystal oscillator (Y1) generates a 50 MHz square wave which is divided by two by a J-K flip-flop (U5) to produce the master 25 MHz clock signal (CLK). CLK gates another J-K flip-flop to produce the 12.5 MHz master clock qualifier signal (CLKQP). To ensure adequate set-up and hold times, CLKQP is delayed by 14 ns to produce CLKQD. The clock signals are reproduced and distributed by low skew clock drivers (U7, U8, U9, U10) to each VASP slot. At each slot is a jumper block where the CLK/CLKQ can be terminated if a VASP board is not inserted in that slot.

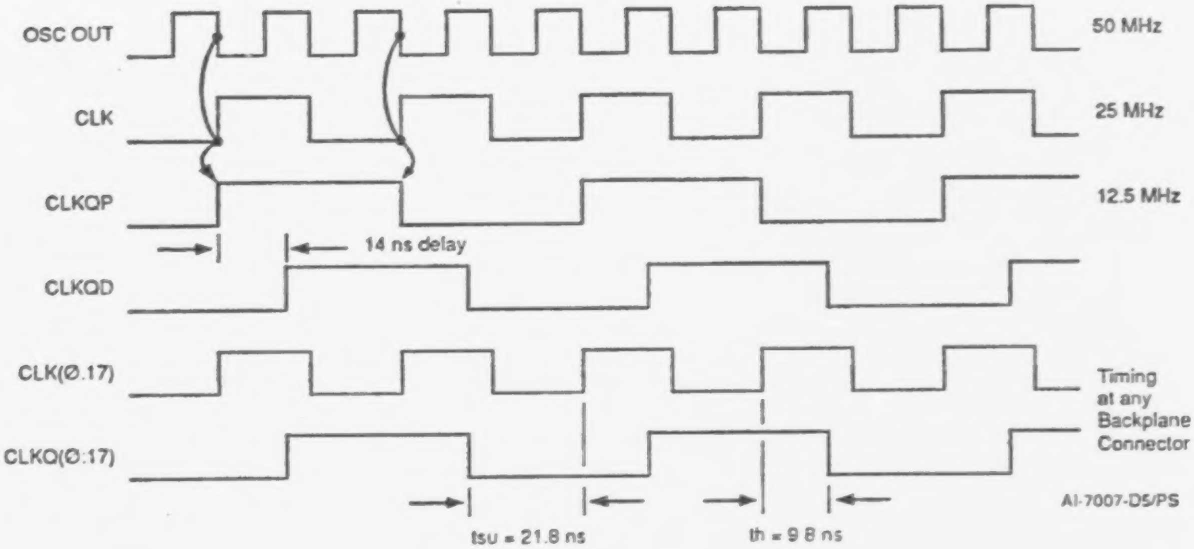

OSC OUT
50 MHz
CLK
25 MHz
CLKQP
12.5 MHz
14 ns delay
CLKQD
CLK(Ø:17)
Timing at any Backplane Connector
CLKQ(Ø:17)
AI-7007-D5/PS
tsu = 21.8 ns
th = 9.8 ns

Figure 7-4 SIGNAL BUS CLOCK GENERATOR TIMING

7.2.5.9 VME Bus Timer Circuit Design

7.2.5.9.1 **Context Description.** The VME Bus Timer monitors the VMEbus activity and drives the VME bus error signal.

7.2.5.9.2 **Functional Description.** The VMEbus Timer starts a timer at the beginning of each VME data transfer, if the timer times out prior to the completion of the transfer the VMEbus data transfer cycle is terminated with a bus error.

7.2.5.9.3 **Physical Description.** The VME Bus Timer monitors the following signals:

a) DS<0:1>/: Data transfer strobes driven by bus master;

b) DTACK/: Data transfer acknowledge driven by slave;

c) SYSRESET/: System reset driven by master or reset switch.

The VME Bus Timer drives the following signals:

a) BERR/: Bus error received by master.

7.2.5.9.4 **Electrical Interface.** The timing of the VME Bus Timer is shown in Figure 7-5. The electrical characteristics of the signals are defined in the VMEbus Specification [R-8].

7.2.5.9.5 **Circuit Operation.** (RTSG Motherboard Subassembly Schematic Drawing 28-0494, Appendix H): A monostable (U2a) is triggered at the falling edge of either data transfer strobe (DSx/) causing the output (Q) to go high. If DTACK/ or SYSRESET/ is not received prior to the monostable timeout, the falling edge of Q will clock the flip-flop (U3) and drive BERR/ via U9. BERR/ will remain asserted until either DSx/ is deasserted or DTACK/ or SYSRESET/ is asserted. If DTACK/ or SYSRESET/ is received the flip-flop (U3) is held in the set state and the monostable (U2a) is cleared.

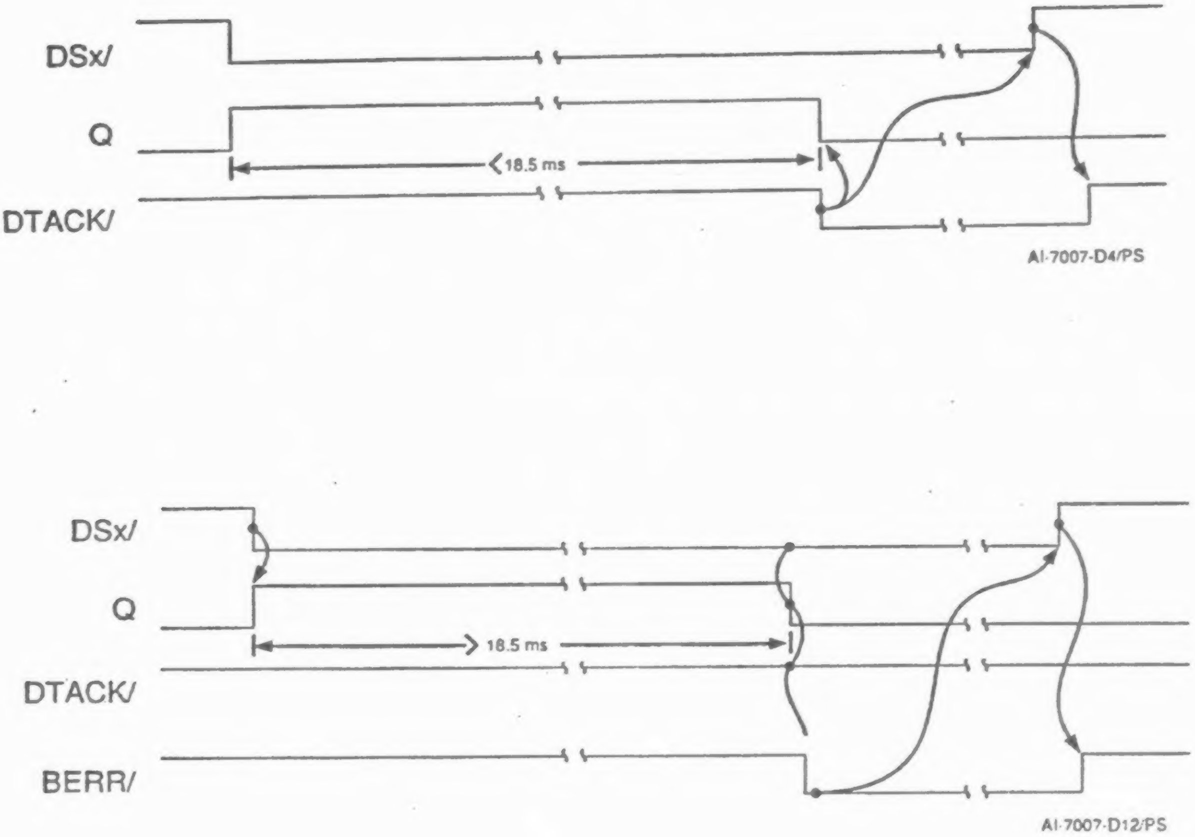

DSx/
Q
18.5 ms
DTACK/
AI-7007-D4/PS
DSx/
Q
18.5 ms
DTACK/
BERR/
AI-7007-D12/PS

Figure 7-5 VME BUS TIMER TIMING WITH: a) NO TIMEOUT and b) TIMEOUT

7.2.5.10 Reset Switch Debouncer Circuit Design

7.2.5.10.1 **Context Description.** The Reset Switch Debouncer monitors the reset switch input and drives the system reset signal.

7.2.5.10.2 **Functional Description.** The Reset Switch Debouncer eliminates fast pulsations on the system reset signal.

7.2.5.10.3 **Physical Description.** The Reset Switch Debouncer monitors the following signals:

a) MANRESET/: Manual reset signal from the Chassis reset switch.

The Reset Switch Debouncer drives the following signals:

a) SYSRESET/: System reset received by all masters and slaves.

7.2.5.10.4 **Electrical Interface.** The Reset Switch Debouncer drives SYSRESET according to the VMEbus Specification [R-8].

7.2.5.10.5 **Circuit Operation.** (RTSG Motherboard Subassembly Schematic Drawing 28-0494, Appendix H): A monostable (U2b) is triggered at the falling edge of MANRESET/ caused by the first closure of the switch contracts causing the output (Q) to go high and drive SYSRESET/ via U4b. Subsequent oscillations at the input of the monostable caused by bouncing contacts will retrigger the monostable if they occur before the period of the monostable oscillation (126 ms) thus filtering the pulsations from the SYSRESET/ signal.

7.2.6 Power Supply

The Power Supply is a custom configured COTS item. The Power Supply is manufactured by Vicor (Westcor) and provided by Mupac along with the Chassis, Cooling Fans and Filter, and Card Cage.

7.2.6.1 Requirements

7.2.6.1.1 **Allocated Requirements.** The requirements allocated to the Power Supply are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.4.4.1.1	Alternating Current Power
3.4.4.3.1	Power Source

7.2.6.2 Interface Definition

7.2.6.2.1 **Mechanical.** The layout of the individual channel modules is shown in Figure 7-6. The physical interconnection of the other subassemblies with the Power Supply are shown in the RTSG Chassis Schematic Drawing SDG442051 and Assembly Drawing G442051 (see Appendix F).

7.2.6.2.2 **Primary Power.** The Power Supply has a power factor of no less than 0.75.

7.2.6.2.3 **DC Power.** The total current drain on the Power Supply with a fully loaded chassis is as follows:

a) +5V @ 92.4 A,
b) +2V @ 15.0 A,
c) +12V @ 2.0 A.
d) -12V @ 0.2 A.

The connectors are specified in the MegaPAC Mechanical and Connection Diagrams [R-9]. The wiring is shown in the RTSG Chassis Subassembly Schematic Drawing SDG442051 (see Appendix F). Note that the 2V output is actually set to and specified at 2.1V.

7.2.6.2.4 **Thermal Sense.** The chassis thermostat is connected across the General Shutdown and Return inputs of each power supply channel. The connectors are specified in the MegaPAC Mechanical and Connection Diagrams [R-9].

7.2.6.2.5 **Voltage Monitor.** The DC output of each channel is connected to a Voltage Monitor LED.

7.2.6.2.6 **Voltage Adjustment.** The +5V and +2V are equipped with ±10% output adjustments.

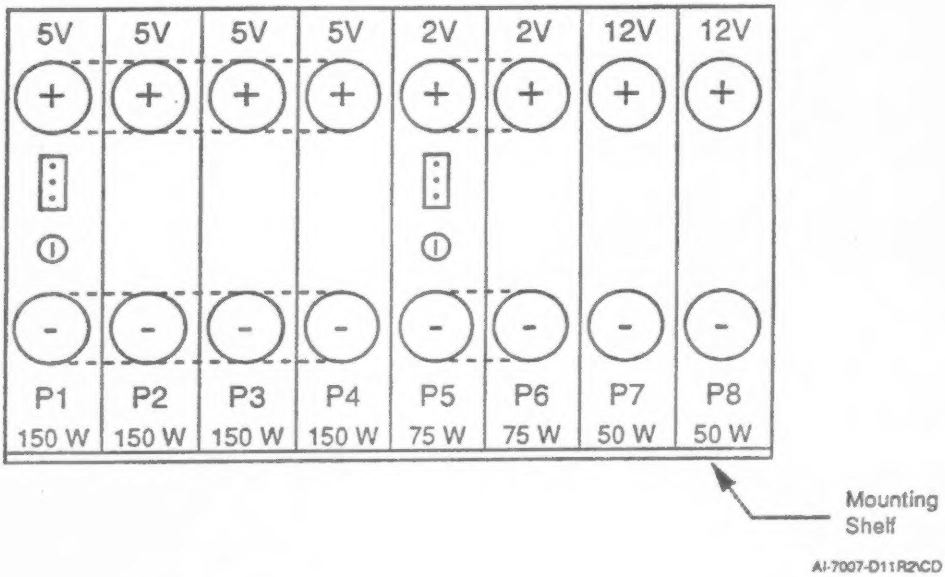

5V
5V
5V
5V
2V
2V
12V
12V
+
-
P1
150 W
P2
150 W
P3
150 W
P4
150 W
P5
75 W
P6
75 W
P7
50 W
P8
50 W
Mounting Shelf
AI-7007-D11R2\CD

Figure 7-6 POWER SUPPLY CONFIGURATION

7.2.6.3 Performance Characteristics

7.2.6.3.1 Primary Power

a) Power Factor: The Power Supply presents a power factor of 0.75 at full load;
b) Power Consumption: With the maximum number of circuit cards and at the minimum specified efficiency of 78% the Power Supply requires a apparent power input of 890 VA or 7.4 A at 120V. The "as-configured" (1 EC, 6 TPM, 3 IOP,) apparent power input is 511 VA or 4.3 A at 120V.

7.2.6.3.2 **DC Power.** The Power Supply provides the following output capabilities:

a) +5V @ 120 A,
b) +2V @ 60A,
c) +12V@ 4.2 A,
d) -12V @ 4.2 A.

The +5V and +2V outputs are equipped with ±10% output adjustment.

7.2.6.4 Physical Characteristics

7.2.6.4.1 **Size.** The approximate size of the Power Supply is 14.0 cm wide x 8.1 cm high x 29.2 cm long.

7.2.6.4.2 **Weight.** The estimated weight of the Power Supply is 3 Kg.

7.2.6.5 Thermal Design

The Power Supply dissipates 157 W with a full complement of circuit cards. The Power Supply is cooled by its own integral fan. The Power Supply contains integral thermal protection which will shut down the Power Supply if its inlet air temperature exceeds 50° C.

7.2.6.6 Electromagnetic Compatibility

The Power Supply meets FCC part 15, Class A requirements for conducted EMI/RFI.

7.2.6.7 Power Distribution and Grounding

7.2.6.7.1 **DC Power.** The DC power distribution is indicated on the RTSG Chassis Subassembly Schematic Drawing SDG442051 (see Appendix F). Each pair of output studs can supply a maximum of 200 W of output power. The +5V output has 4 pairs of studs; the +2V output has 2 pairs of studs, and the remainder have 1 pair of studs each.

7.2.6.8 Reliability

Reliability data has been provided by Westcor for the DC/DC convertors in the Power Supply. The eight DC/DC converters, which comprise the majority of the Power Supply components, are rated at a combined MTBF of 87,500 hours at 35° C operation.

7.2.6.9 Safety

The Power Supply is UL and CSA approved. The exposed metallic components are connected to the AC primary power safety ground.

7.2.7 Chassis

The Chassis is a modified COTS item. It is provided by Mupac along with the Cooling Fans and Filter, Card Cage, and Power Supply.

7.2.7.1 Requirements

7.2.7.1.1 **Allocated Requirements.** The requirements allocated to the Chassis are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.4.4.1.1	Alternating Current Power
3.4.4.2.1	Cables
3.4.4.2.2	Connectors
3.4.4.3.1	Power Source

7.2.7.2 Interface Definition

7.2.7.2.1 **Mechanical.** The mechanical layout and dimensions (inch) of the chassis are shown in Figures 7-7, 7-8, 7-9, and 7-10. These figures are derived from the Chassis specification supplied to Mupac.

7.2.7.2.2 **DC Power.** The wiring is shown in the RTSG Chassis Subassembly Schematic Drawing SDG442051 (see Appendix F).

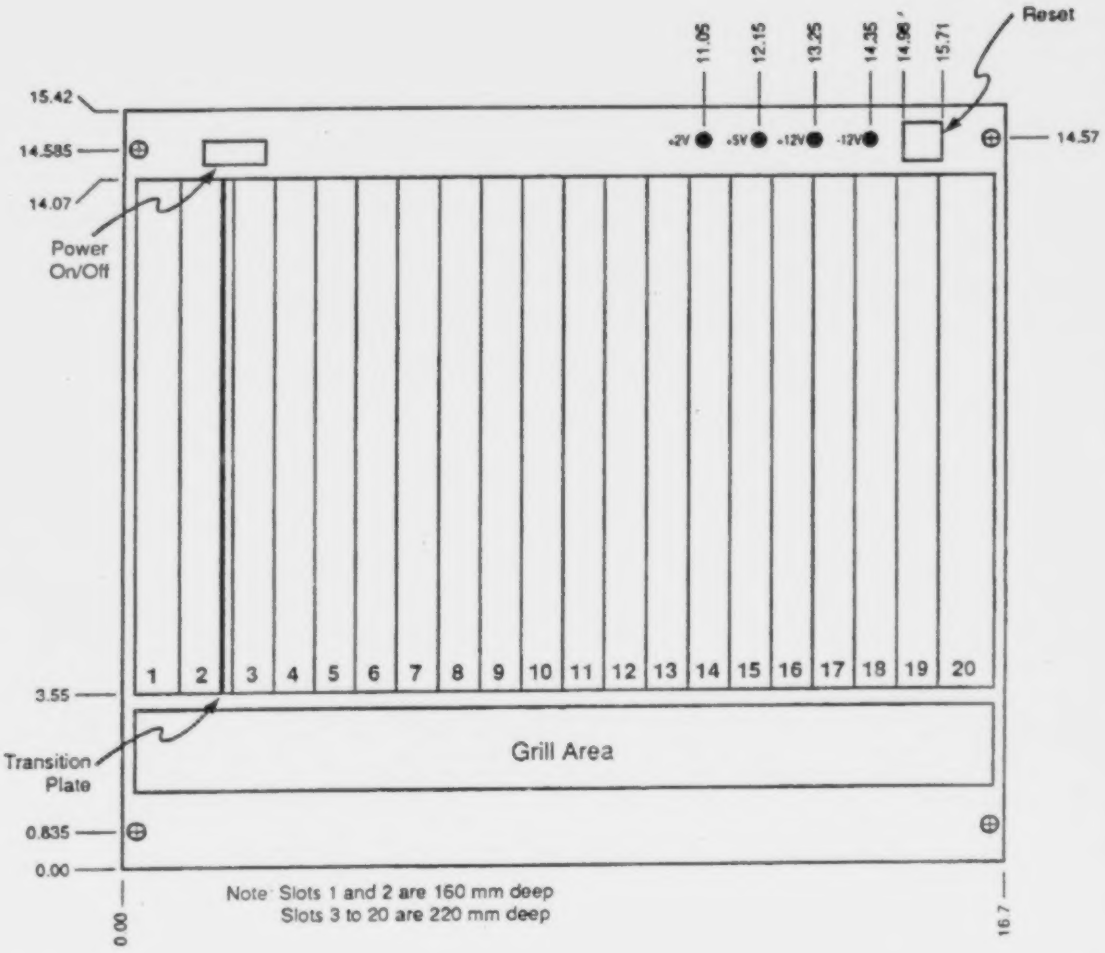
7.2.7.2.3 **Voltage Monitor.** The Voltage Monitor provides the following displays:

a) +5V,
b) +2V,
c) +12V,
d) -12V.

7.2.7.2.4 **Reset Switch.** The reset switch is protected from inadvertent operation by a moveable cover.

7.2.7.2.5 **Power Switch.** The power switch is protected from inadvertent operation by a hood.

7.2.7.2.6 **Interior Access.** The rear interior of the chassis is accessible via the rear panel connector plate.


11.05
12.15
13.25
14.35
14.98
15.71
Reset
15.42
14.585
14.07
14.57
+2V
+5V
+12V
-12V
Power On/Off
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
3.55
Grill Area
Transition Plate
0.835
0.00
0.00
16.7

Note: Slots 1 and 2 are 160 mm deep
Slots 3 to 20 are 220 mm deep

AI-7007-D7R3CD

Figure 7-7 CHASSIS BEZEL FRONT VIEW. (Dimensions in inches)

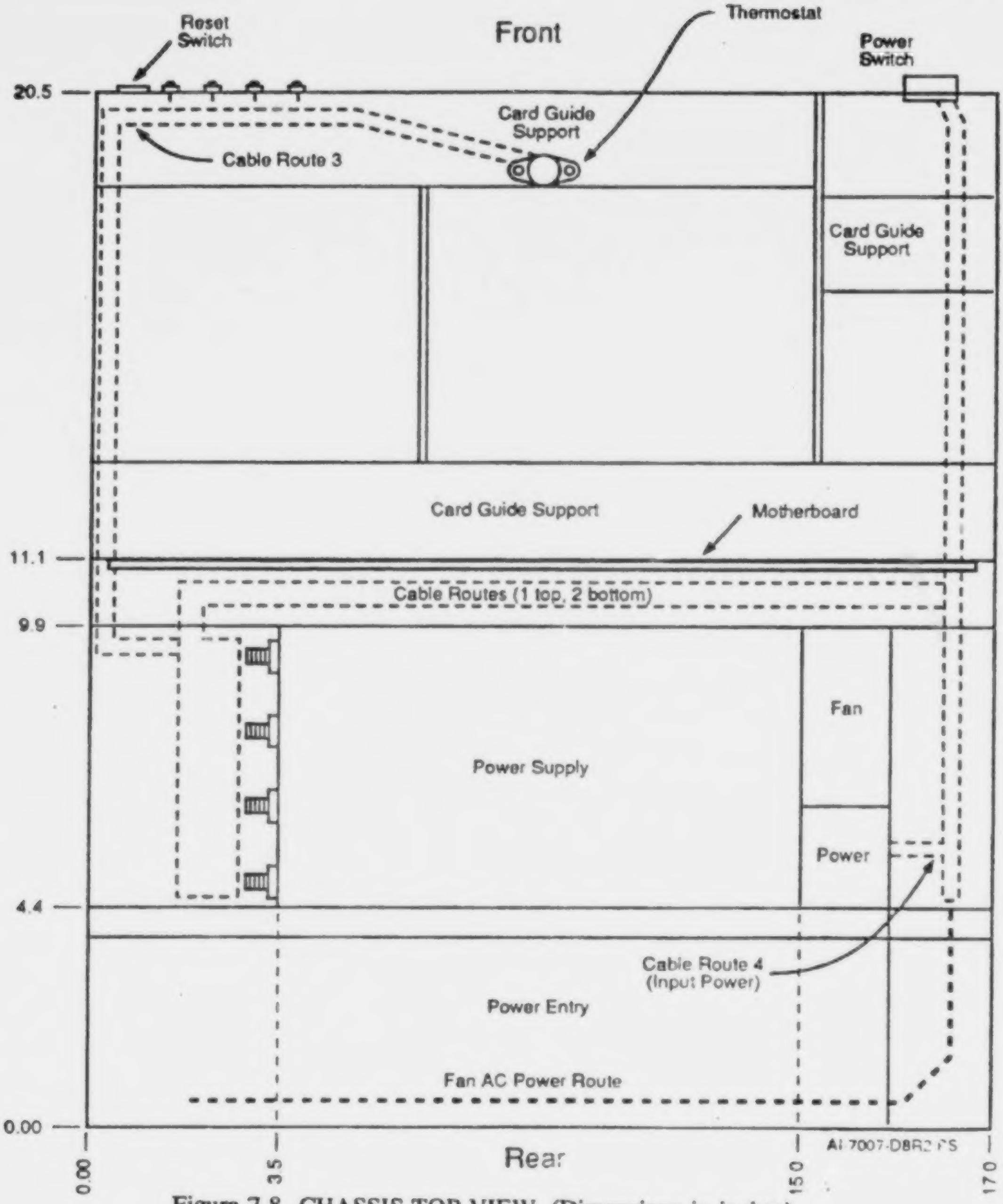

Reset Switch
Front
Thermostat
Power Switch
20.5
Card Guide Support
Cable Route 3
Card Guide Support
Card Guide Support
Motherboard
11.1
Cable Routes (1 top, 2 bottom)
9.9
Fan
Power Supply
Power
4.4
Cable Route 4 (Input Power)
Power Entry
Fan AC Power Route
0.00
Rear
AI-7007-DBR2 FS
0.80
3.5
15.0
17.0

Figure 7-8 CHASSIS TOP VIEW. (Dimensions in inches)

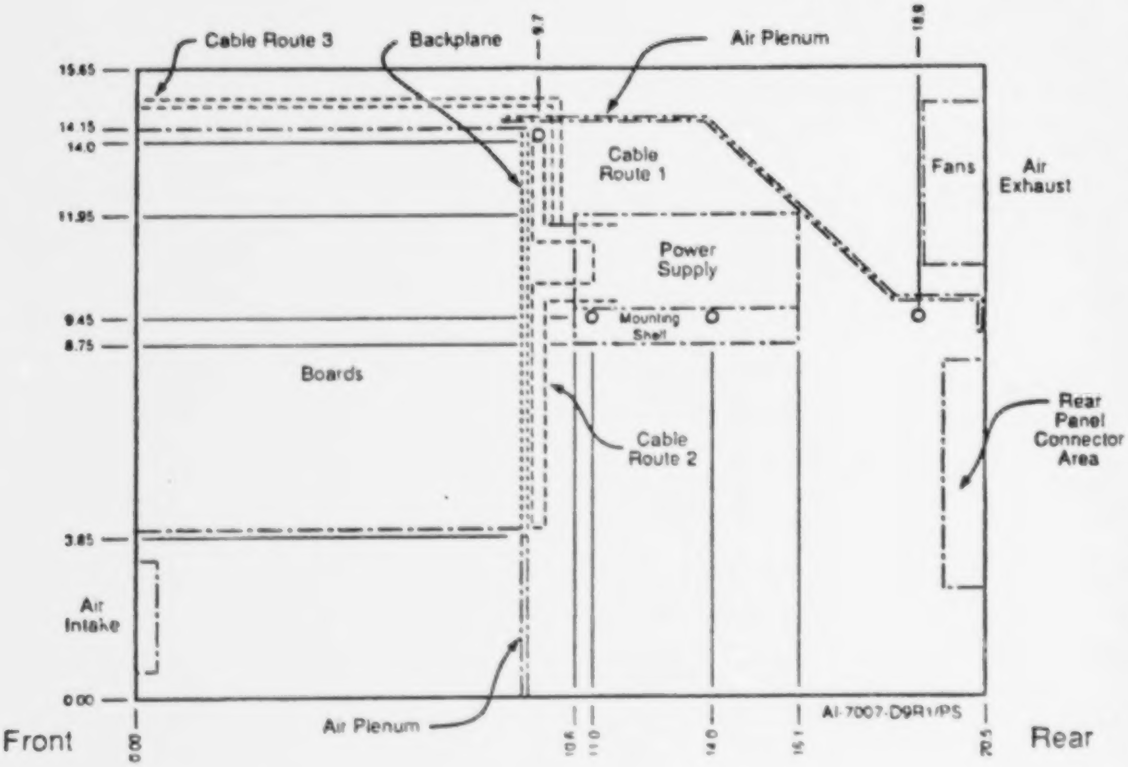
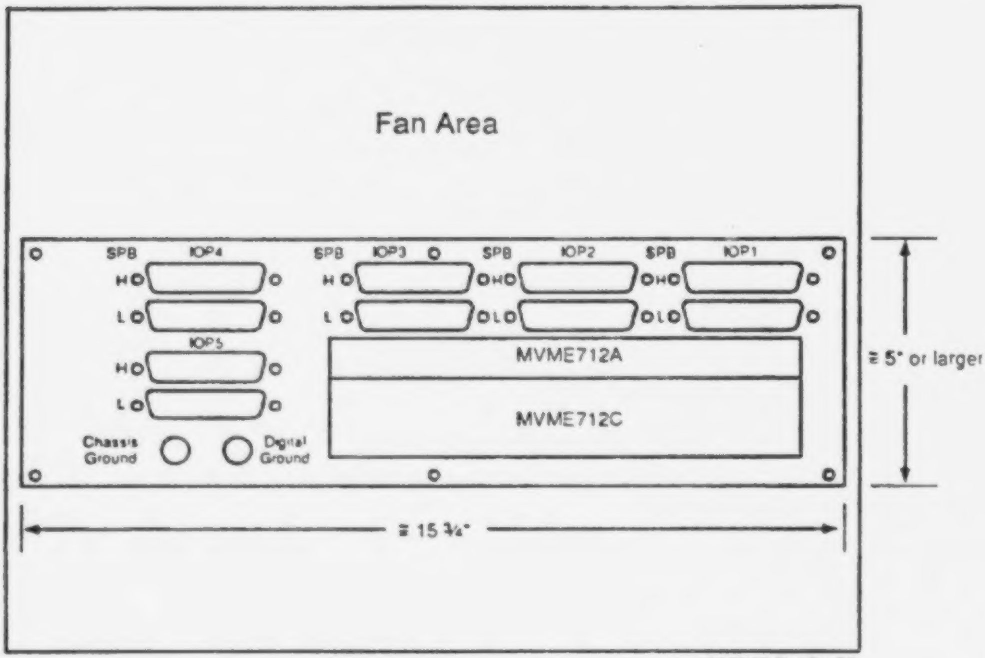

Cable Route 3
Backplane
9.7
Air Plenum
18.8
15.65
14.15
14.0
11.95
9.45
8.75
3.85
0.00
Cable Route 1
Fans
Air Exhaust
Power Supply
Mounting Shelf
Boards
Cable Route 2
Rear Panel Connector Area
Air Intake
Air Plenum
Front
0.0
10.6
11.0
14.0
16.1
AI-7007-D9R1/PS
Rear

Figure 7-9 CHASSIS SIDE VIEW. (Dimensions in inches)


Fan Area
SPB
IOP4
SPB
IOP3
SPB
IOP2
SPB
IOP1
H
L
IOP5
H
L
MVME712A
MVME712C
Chassis Ground
Digital Ground
≅ 5" or larger
≅ 15 ¾"

AI-7007-D10R1/PS

Figure 7-10 REAR PANEL CONNECTOR AREA. (Dimensions in inches)

7.2.7.3 Performance Characteristics

7.2.7.3.1 **Primary Power.** The AC input is equipped with 12 A fuses. The total apparent power input for a fully loaded chassis is 943 VA or 7.9 A at 120V. The "as-configured" (1 EC, 6 TPM, 3 IOP,) total apparent power input is 565 VA or 4.7 A at 120V. Note that the input power to the chassis includes the Power Supply (5.2.7.3.1) and the Cooling Fans (5.2.9.3.2).

7.2.7.3.2 **Thermal Sense.** The thermostat is rated at 43°C ± 3°C.

7.2.7.4 Physical Characteristics

7.2.7.4.1 **Size.** The approximate size of the Chassis is 43.2 cm wide x 39.2 cm high x 52.1 cm long.

7.2.7.4.2 **Weight.** The estimated weight of the Chassis alone is 6 kg.

The weight of the Chassis complete with various items is as follows:

a) Chassis, Card Cage, Power Supply, Backplane, Cooling Fans and Filter, IOP Adapters, MVME712A, MVME712C, MVME147P2 (installation weight without circuit boards installed): 17 kg;
b) As in (a) above with as configured board set (1 EC, 3 IOP, 6 TPM): 27 kg;
c) As in (a) above with maximum number of boards: 37 kg.

7.2.7.5 Thermal Design

The Chassis has two separate ventilation systems; one for the Power Supply and one for the Card Cage. The Power Supply has its own fan and thermal shutdown features.

The Chassis contains a thermostat rated at 43 ± 3°C. This ensures that the Power Supply will not be disabled until the Card Cage exhaust temperature reaches 40°C minimum, which is the sum of the specified maximum room temperature (30° C) and the designed maximum temperature rise through the card cage (10° C). It also ensures that the power will be cut-off before 46°C, which is less than the sum of the maximum VASP operating temperature (40°C) and the maximum temperature rise through the card cage (10°C).

The VASP boards are tested by the manufacturer for maximum operating temperature by installing them in a chassis in a thermal chamber and raising the air temperature in the chamber to the specified maximum of 40°C. The test chassis is designed with the same air flow and temperature rise (10°C) characteristics as the SBR-SPD chassis.

7.2.7.6 Electromagnetic Compatibility

The leakage of electromagnetic energy from the ventilation openings is minimized by the ventilation slots and fan finger guards.

The DC supply lines to the Backplane are each twisted with a return line to reduce the radiated magnetic field.

The AC wiring is routed away from the DC wiring to reduce coupling between them (see Appendix F, RTSG Chassis Subassembly Schematic Drawing, SDG442051).

7.2.7.7 Power Distribution and Grounding

7.2.7.7.1 **DC Power.** The Wiring from the power supply to the Backplane is shown on the RTSG Chassis Subassembly Schematic Drawing SDG442051 (see Appendix F). The wiring was designed with the following capabilities:

a) +5V: 13 #14 AWG conductors capable of a total of 123 A for a 10°C temperature rise above ambient;

b) +2V: 8 #12 AWG conductors capable of a total of 104 A for a 10°C temperature rise above ambient;

c) 12V: 1 #14 AWG conductor each, capable of a total of 9.5 A for a 10°C temperature rise above ambient;

d) GND: all return lines matched to supply lines.

The sense signals returned from the Backplane to the Power Supply allow compensation for voltage drops on the +5V and +2V wiring.

7.2.8 Cooling Fans and Filter

The Cooling Fans and Filter are COTS items. They are supplied by Mupac along with the Chassis, Card Cage, and Power Supply.

7.2.8.1 Interface Definition

7.2.8.1.1 **Mechanical.** Three fans are used to provide the required air flow.

7.2.8.1.2 **Primary Power.** The Fans operate from the Chassis AC input power (115 V/60 Hz).

7.2.8.2 Performance Characteristics

7.2.8.2.1 **Air Flow.** The fans are rated at 300 CFM.

7.2.8.2.2 **Power Consumption.** Each fan consumes approximately 18 W for a total of 54 W or 0.5 A at 115 V.

7.2.8.3 Physical Characteristics

7.2.8.3.1 **Size.** The approximate size of each fan is 10.5 cm square x 3.8 cm thick.

7.2.8.3.2 **Weight.** The estimated weight of each fan is 0.5 Kg for a total of 1.5 Kg.

7.2.8.4 Thermal Design

The total power dissipated by the maximum number of circuit cards and the backplane is approximately 600 W. The temperature rise in a cubic foot of air at sea level at 25° C is given as 1.78°C per minute per watt. The total fan rating for the specified temperature rise of 10°C within the chassis is calculated as follows:

$600 \cdot 1.78/10 = 107$ CFM.

To compensate for hot spots, uneven air flow, backpressure, higher inlet temperature (30° C), and air flow restrictions, the base requirement is doubled to 214 CFM.

The vendor is providing 300 CFM of total fan airflow rating for additional margin.

7.2.8.5 Reliability

A survey of fan manufacturer's specifications indicates that fans of this type have typical life expectancies of 30,000 + hours at the maximum operating temperature of the RTSG (40° C).

7.2.9 IOP Adapter

The IOP Adapter is a developmental item. The bare PCB is manufactured by Circuit Graphics.

7.2.9.1 Requirements

7.2.9.1.1 **Allocated Requirements.** The requirements allocated to the IOP Adapter are shown below. All requirements are traced to the Testbed Specification [A-2] unless indicated by [I/F], in which case they trace to the DSP/Testbed Interface Specification [A-1].

Number	Title
3.4.3.2.1	DSP Hardware Interface
5.2.2.2.1	ESP/RSS Hardware Interface
3.3.1.1.2[I/F]	Hardware Acknowledge

7.2.9.2 Interface Definition

7.2.9.2.1 **Mechanical.** The mechanical layout of the IOP Adapter is shown on the IOP Adapter Subassembly Fabrication Drawing 28-0449 and Assembly Drawing 28-0447 (see Appendix H.

7.2.9.2.2 **Synchronous Parallel Bus.** The IOP Adapter provides a transition from the Backplane Metral Connector and the rear panel "D" connector. The IOP Adapter permits the crossover wiring of the SPB control lines as shown on the IOP Adapter Subassembly Schematic Drawing 28-0446 (see Appendix H). This allows the use of a universal IOP Adapter for both transmit and receive configurations of the SPB.

7.2.9.3 Performance Characteristics

The IOP Adapter converts the SPB signals from an IOP backplane slot rear plug-up connector to two 50 position ribbon cable headers. This permits a ribbon cable to run from the rear panel connectors to the IOP Adapter.

7.2.9.4 Physical Characteristics

7.2.9.4.1 **Size.** The IOP Adapter is approximately 9.4 cm x 6.4 cm x 1.5 cm.

7.2.9.4.2 **Weight.** The IOP Adapter weight is estimated at 0.12 Kg.

7.3 TRANSPORTABILITY, PACKAGING, HANDLING

7.3.1 Transportability

The RTSG equipment is designed for permanent installation at the customer's site and is not designed for frequent transportation between sites. The equipment is mounted on a rack which has casters; this permits some movement within the interior of the same building without disassembly.

7.3.2 Packaging

The RTSG equipment should be packaged for shipping to the customer in the same containers in which it was received from the vendor for maximum protection (corrugated cardboard box with polystyrene foam cushioning). Equivalent or better packaging may be used if the original containers are not available or are not reusable.

The Chassis must be removed from the rack and the rack disassembled for shipping. The Chassis, EC, IOPS, and TPMs should be removed from the Card Cage and packaged in their original containers. The remaining components assembled into the chassis should be left in the chassis permanently.

7.3.3 Handling

The chassis may require two people to lift.

Any commercial mode of transport may be used so long as it provides protection from moisture, shock, extreme pressure or temperature variations, and strong electromagnetic or electrostatic fields.

7.4 SUPPORT EQUIPMENT REQUIREMENTS

These items are in addition to any deliverable equipment which may be used in a support role.

7.4.1 Development and Test

The following items are required for development and test of the RTSG:

a) Corporate computing facilities (printers, terminals, network, computers);
b) Mechanical and electrical assembly and inspection equipment;
c) VASP test set;

d) Oscilloscope, volt-ohm-meter, logic analyzer, VME bus analyzer, thermometer;
e) Thermal chamber;
f) VME extender card;
g) VMEbus Analyzer Card.

7.4.2 Installation

The following items are required for installation of the RTSG:

a) Standard tool-kit (MacDonald Dettwiler issued).

7.4.3 Maintenance

The following items are required for maintenance of the RTSG:

a) Clean, dry , lint-free cloths for cleaning chassis exteriors;

b) Mild detergent solution for cleaning air filter.

SECTION EIGHT

RSS/CONTROLLER INTERFACE DETAILED DESIGN

8.1 INTRODUCTION

This section describes the interface design between the RSS and the Controller. Both the RSS and the Controller hardware are based on identical computer platforms (Sun SPARCstations) where communicate with each other over an IEEE 802.3 standard Local Area Network interface. The software interface design between the RSS and the Controller is described at a high level in the following subsections with references to Appendices A and D for detailed design information. The interface messages and protocols designs are described in the constituent software packages of the RSS Interface CSC. The RSS generated products file formats which are used by the Controller during data analysis and simulation run are described in the Signal Generation Products CSU. The interface diagram between the RSS and the Controller is shown in Figure 8-1.

8.2 RSS INTERFACE (RIF) CSC

The RIF CSC provides the definition of the interface between the Controller and the RSS CSCs for signal generation and data playback activities. This defines all the messages, message content and format and the protocol used in sending messages between the Controller and RSS functional areas. This does not include any application software to actually send and receive the messages.

The RIF CSC described in this detailed design document differs from the RIF CSC described in the preliminary design document (Document R-1) in the following areas:

a) the RIF CSC is no longer decomposed into the RGI and RPI sub-level CSCs (these sub-level CSCs are instead CSUs);

b) the RIF CSC no longer includes the Generation Server and Simulation Server tasks to communicate with the RSS (this is now allocated, along with the corresponding requirements, as part of the System Monitor and Control "SMC" CSC).

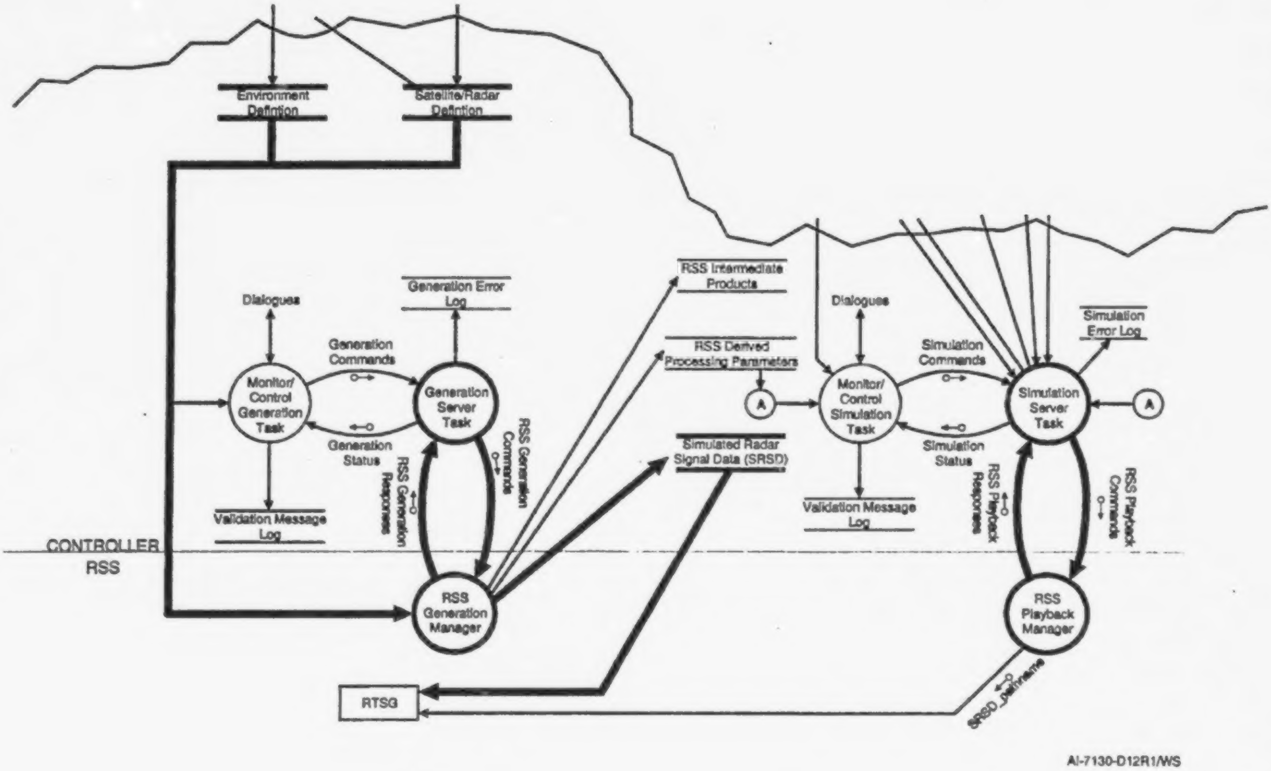

Environment Definition
Satellite/Radar Definition
Dialogues
Generation Error Log
Generation Commands
Monitor/ Control Generation Task
Generation Server Task
Generation Status
RSS Generation Commands
RSS Generation Responses
Validation Message Log
CONTROLLER
RSS
RSS Generation Manager
RTSG
RSS Intermediate Products
RSS Derived Processing Parameters
A
Simulated Radar Signal Data (SRSD)
Dialogues
Monitor/ Control Simulation Task
Simulation Commands
Simulation Status
Simulation Server Task
Simulation Error Log
A
Validation Message Log
RSS Playback Responses
RSS Playback Commands
RSS Playback Manager
SRSD_pathname
AI-7130-D12R1/WS

Figure 8-1 RSS/Controller Interface Diagram

The RIF CSC consists of the following CSUs:

- RSS Generation Interface (CRGI) CSU
- RSS Playback Interface (CRPI) CSU

8.2.1 RSS Generation Interface (CRGI) CSU

The purpose of the CRGI CSU is to provide the definition of the interface between the Controller and RSS for signal generation activities. This defines all the messages (message content and format) and the protocol used in sending messages.

8.2.1.1 Requirements

The following requirements have been allocated from the Testbed specification (Document A-2) to this CSU:

Number	Title
3.7.1.1.1	Interface Requirements
3.7.1.1.5	Signal Generation Parameters
4.1.1.9.1	Interface Control
4.2.2.1.1	Controller/RSS Software Interface

8.2.1.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of local data files or data base, and limitations for use of each CSU are explained as applicable in the following section:

a) Section D1.4 - provides a complete description of the interface specification.

8.2.2 RSS Playback Interface (CRPI) CSU

The purpose of the CRPI CSU is to provide the definition of the interface between the Controller and RSS for controlling RSS data playback during simulation run activities. This defines all the messages (message content and format) and the protocol used in sending messages.

8.2.2.1 Requirements

The following requirements have been allocated from the Testbed specification (Document A-2) to this CSU:

Number	Title
3.7.1.1.1	Interface Requirements
4.1.1.9.1	Interface Control
4.2.2.1.1	Controller/RSS Software Interface

8.2.2.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of local data files or data base, and limitations for use of each CSU are explained as applicable in the following section:

a) Section D1.5 - provides a complete description of the interface specification.

8.2.3 Signal Generation Products (CGEP) CSU

The purpose of the CGEP CSU is to provide the definition of the products generated by the RSS during a signal generation activity which will be used by the Controller software during simulation run and data analysis. This consists of a separate definition for each of the following product files (note that the SRSD files are RSS internal files which are generated by the RSS software and used by the RSS software for simulation run data playback only, if it is desirable to analyze the simulated radar signals, the ESP input may be captured during a simulation run):

a) RSS Intermediate Products as follows:

- Clutter signal
- External noise signal
- Jammer signal
- Target signal
- Composite signal at antenna
- Post nonlinearity signal
- IF-filter output signal
- Post synchronous detector
- A/D output signal
- Antenna pattern
- IF-filter response

b) RSS derived processing parameters

8.2.3.1 Requirements

The following requirements have been allocated from the Testbed Specification (Document A-2) to this CSU:

Number	Title
3.3.3.1.1	Data Logging
4.1.1.9.2	Data Logging

8.2.3.2 Design Description

The input/output data elements, local data elements, interrupts and signals, algorithms, error handling, data conversion, use of other elements, logic flow, data structures, structures of local data files or data base, and limitations for use of each CSU are explained as applicable in the following sections:

a) Section A5.2.3 - provides a description of the file characteristics;

b) Section D1.12 - provides a complete description of the interface specification.

SECTION NINE

REQUIREMENTS TRACEABILITY

The following tables show the requirements traceability of the Testbed Detailed Design to the Testbed Specification and the DSP/Testbed Interface Specification:

a) Table 9-1 shows allocation of Testbed subsystem software requirements, Section 3 of the Testbed specification (Document A-2);

b) Table 9-2 shows allocation of Testbed Controller software requirements, Section 4 of the Testbed specification (Document A-2);

c) Table 9-3 shows allocation of RSS software requirements, Section 5 of the Testbed specification (Document A-2);

d) Table 9-4 shows allocation of DSP/Testbed interface software requirements from the DSP/Testbed interface specification (Document A-1);

e) Table 9-5 shows allocation of Testbed subsystem hardware requirements, Section 3 of the Testbed specification (Document A-2);

f) Table 9-6 shows allocation of Testbed Controller hardware requirements, Section 4 of the Testbed specification (Document A-2);

g) Table 9-7 shows allocation of RSS hardware requirements, Section 5 of the Testbed specification (Document A-2);

h) Table 9-8 shows allocation of DSP/Testbed interface hardware requirements from the DSP/Testbed interface specification (Document A-1).

Table 9-1 Testbed Software Requirements Traceability

Spec #	Requirement Title	CSC Name	CSU Name
[A-2]	3.1.2.0.1 External Events Recognized	SGD	CRDH
		SGD	CENH
		SRD	CSDH
		SMC	CGEH
		SMC	CGET
		SMC	CSIH
		SMC	CSIT
		DAN	CDAT
[A-2]	3.2.1.0.1 Define Radar Signal Generation Response	SGD	CRDH
		SGD	CENH
[A-2]	3.2.1.0.2 Invoke Radar Signal Generation Response	SMC	CGEH
		SMC	CGET
[A-2]	3.2.1.0.3 Define Simulation Run Response	SRD	CSDH
[A-2]	3.2.1.0.4 Invoke Simulation Run Response	SMC	CSIH
		SMC	CSIT
[A-2]	3.2.1.0.5 Capture DSP Simulation Results Response	SMC	CSIT
[A-2]	3.2.1.0.6 Analyze Results Response	DAN	CDAT
[A-2]	3.3.1.0.1 Multiple User Operation	SGD	CRDH
		SGD	CENH
		SRD	CSDH
		DAN	CDAT

Spec #	Requirement Title	CSC Name	CSU Name
[A-2]	3.3.1.0.2 Operation Without DSP	SGD SGD SRD DAN	CDRH CENH CSDH CDAT
[A-2]	3.3.2.0.1 Define Signal Generation Parameters	SGD SGD	CRDH CENH
[A-2]	3.3.2.0.2 Generate Signal Data	SSGM	SGEN
[A-2]	3.3.2.0.3 Define Simulation Run Parameters	SRD	CSDH
[A-2]	3.3.2.0.4 Playback Signal Data	SPMT	SPMT
[A-2]	3.3.2.0.5 Effects Modelled	SSGM SSGM	SBIT SBMT
[A-2]	3.3.2.0.6 Baseline Definition	SGD SRD	CRDH CSDH
[A-2]	3.3.2.0.7 Signal Processing Reconfigurability Definition	SRD	CSDH
[A-2]	3.3.3.1.1 Data Logging	DMT DMT	CGEP CSIP
[A-2]	3.3.3.2.2 Performance Metrics Display	SMC	CSIH
[A-2]	3.3.4.0.1 Analysis Capability	DAN	CDAT
[A-2]	3.3.4.0.2 Hard Copy Output	DAN	CDAT
[A-2]	3.3.4.0.3 Parameter Estimation	DAN	CDAT
[A-2]	3.4.1.1.2 Operator Multitasking	SGD SGD SRD DAN	CRDH CENH CSDH CDAT
[A-2]	3.4.1.1.3 Default Values	SGD SGD SRD	CRDH CENH CSDH

Spec #	Requirement Title	CSC Name	CSU Name
[A-2]	3.4.1.1.4 Units of Measure	SGD SGD SRD SMC SMC	CRDH CENH CSDH CGEH CSIH
[A-2]	3.4.1.1.5 Positive Confirmation	SGD SGD SRD	CRDH CENH CSDH
[A-2]	3.4.1.2.1 Signal Generation Parameter Hierarchy	SGD	CRDH
[A-2]	3.4.1.2.3 Data Segment Interval	SGD	CRDH
[A-2]	3.4.1.2.4 CT Interval	SGD	CRDH
[A-2]	3.4.1.2.5 Dwell Interval	SGD	CRDH
[A-2]	3.4.1.2.6 NCI Interval	SGD	CRDH
[A-2]	3.4.1.2.7 CP Interval	SGD	CRDH
[A-2]	3.4.1.2.8 Environment Definition	SGD	CENH
[A-2]	3.4.1.2.9 Data Capture Options	SGD	CRDH
[A-2]	3.4.1.2.10 Signal Generation Data Management	SGD SGD	CRDH CENH
[A-2]	3.4.1.2.11 Satellite/Radar Parameter Validation	UTL	CVAL
[A-2]	3.4.1.2.12 Environment Parameter Validation	UTL	CVAL
[A-2]	3.4.1.3.1 Simulation Run Parameter Hierarchy	SRD	CSDH
[A-2]	3.4.1.3.2 Data Segment ID	SRD	CSDH
[A-2]	3.4.1.3.3 RSP Configuration	SRD	CSDH
[A-2]	3.4.1.3.6 ESP Configuration	SRD	CSDH
[A-2]	3.4.1.3.7 DSP Processing Parameters	SRD	CSDH
[A-2]	3.4.1.3.8 Data Capture Options	SRD	CSDH
[A-2]	3.4.1.3.9 Simulation Run Data Management	SRD	CSDH
[A-2]	3.4.1.3.10 Simulation Run Parameter Validation	UTL	CVAL

Spec #	Requirement Title	CSC Name	CSU Name
[A-2]	3.4.1.4.1 Signal Generation Model Definition	SGD	CRDH
[A-2]	3.4.1.5.1 Signal Generation HMI	SMC	CGEH
[A-2]	3.4.1.6.1 Simulation Run HMI	SMC	CSIH
[A-2]	3.4.1.7.1 SPD User Access	UTL	CSYS
[A-2]	3.4.1.7.2 SPD System Manager Access	UTL	CSYS
[A-2]	3.4.1.7.3 Offline Access	SGD SGD SRD SMC SMC	CRDH CENH CSDH CGEH CSIH
[A-2]	3.4.1.8.1 Select Data Sets	DAN	CDAT
[A-2]	3.4.3.1.1 DSP Software Interface	SMC	CSIT
[A-2]	3.5.1.1.1 Real-Time Operation	SMC	CSIT
[A-2]	3.5.1.1.2 Scope of Real-Time Performance Demonstration	SMC	CSIT
[A-2]	3.6.1.2.1 Software Reconfigurability	SRD	CSDH
[A-2]	3.6.6.0.1 Access Control	UTL	CSYS
[A-2]	3.7.1.1.1 Interface Requirements	RIF RIF RIF	CRGI CRPI CDSI
[A-2]	3.7.1.1.5 Signal Generation Parameters	RIF	CRGI

Table 9-2 Controller Software Requirements Traceability

Spec #	Requirement Title	CSC Name	CSU Name
[A-2]	4.1.1.1.1 Operation Capabilities	SGD SGD SRD DAN	CRDH CENH CSDH CDAT
[A-2]	4.1.1.1.2 HMI Capabilities	SGD SGD SRD DAN	CRDH CENH CSDH CDAT
[A-2]	4.1.1.2.1 HMI For Signal Generation Definition	SGD SGD	CRDH CENH
[A-2]	4.1.1.2.2 Signal Generation Validation	UTL	CVAL
[A-2]	4.1.1.3.1 HMI For Simulation Run Definition	SRD	CSDH
[A-2]	4.1.1.3.2 Simulation Run Validation	UTL	CVAL
[A-2]	4.1.1.4.1 HMI For Signal Generation Models	SGD	CRDH
[A-2]	4.1.1.5.1 HMI For Invoke Signal Generation	SMC	CGEH
[A-2]	4.1.1.6.1 HMI For Invoke Simulation Run	SMC	CSIH
[A-2]	4.1.1.7.1 HMI For Perform System Control	UTL	CSYS
[A-2]	4.1.1.7.4 Control of Activities	SMC SMC SMC SMC	CGEH CGET CSIH CSIT
[A-2]	4.1.1.7.5 Maintain Errorlog	SGD SGD SRD SMC SMC SMC SMC UTL	CRDH CENH CSDH CGEH CGET CSIH CSIT CERR
[A-2]	4.1.1.8.1 HMI For Analyze Simulation Results	DAN	CDAT

Spec #	Requirement Title	CSC Name	CSU Name
[A-2]	4.1.1.8.1 HMI For Analyze Simulation Results	DAN	CDAT
[A-2]	4.1.1.8.2 Analysis Capability	DAN	CDAT
[A-2]	4.1.1.9.1 Interface Control	RIF RIF RIF SMC SMC	CRGI CRPI CDSI CGET CSIT
[A-2]	4.1.1.9.2 Data Logging	DMT DMT	CGEP CSIP
[A-2]	4.1.1.10.1 Interface Control	SMC	CSIT
[A-2]	4.1.1.10.2 Data Logging	DMT	CSIP
[A-2]	4.1.1.10.3 Simulation Metrics	SMC	CSIT
[A-2]	4.2.1.1.1 Controller/HMI Software Interface	SGD SGD SRD SMC SMC SMC	CRDH CENH CSDH CGEH CSIH CSIT
[A-2]	4.2.2.1.1 Controller/RSS Software Interface	RIF RIF DIF DAN	CRGI CRPI CDSI CDAT
[A-2]	4.2.3.0.1 Controller/DSP Interface	SMC	CSIT
[A-2]	4.3.1.0.1 Real-Time Demonstration	SMC	CSIT

Table 9-3 RSS Software Requirements Traceability

Spec #	Requirement Title	CSC Name	CSU Name
[A-2]	5.1.0.0.1 Signal Generation Capabililty	SSGM	SSGM
	5.1.1:1.1 Inter burst changes		SRUN
	5.1.1.1.2 Inter dwell changes		SRUN
	5.1.1.2.1 Pulse Modulation		SRUN
	5.1.1.2.2 Waveform type		SRUN
	5.1.1.10.2 Receiver Channels		SRUN
	5.3.0.0.2 Signal Generation		SSGM
[A-2]	5.1.0.0.3 Effects Modelled (partial)	SBIT	SBIT
	5.1.1.3.1 Earth Rotation (partial)		SBIT
	5.1.1.3.2 Spacecraft Simulation (partial)		SBIT
	5.1.1.4.1 Earth Clutter Map (partial)		SBIT
	5.1.1.4.2 Earth Clutter Models (partial)		SBIT
	5.1.1.4.3 Volumetric Clutter Map (partial)		SBIT
	5.1.1.4.4 Volumetric Clutter Models (partial)		SBIT
	5.1.1.5.1 Lower Atmospheric Propagation		SBIT
	5.1.1.5.2 Ionospheric Propagation		SBIT
	5.1.1.6.1 Main Antenna		SBIT
	5.1.1.6.2 Auxiliaries		SBIT
	5.1.1.9.2 Auxiliary Re-use		SBIT
[A-2]	5.1.0.0.3 Effects Modelled (partial)	SBMT	SBMT
	5.1.1.3.1 Earth Rotation (partial)		SBMT
	5.1.1.3.2 Spacecraft Simulation (partial)		SBMT
	5.1.1.4.1 Earth Clutter Map (partial)		SBMT
	5.1.1.4.2 Earth Clutter Models (partial)		SBMT
	5.1.1.4.3 Volumetric Clutter Map (partial)		SBMT
	5.1.1.4.4 Volumetric Clutter Models (partial)		SBMT
	5.1.1.7.1 Jammer Types		SBMT
	5.1.1.7.2 Jammer Distortion		SBMT
	5.1.1.8.1 Target Models		SBMT
	5.1.1.9.1 Noise Models		SBMT
	5.1.1.10.1 Quadratic Phase Modulatioin		SBMT
	5.1.1.10.3 Receiver Front-end		SBMT
	5.1.2.0.1 Sample Characteristics		SBMT
[A-2]	5.1.0.0.2 Signal Playback Capability	SPMT	SPMT

Spec #	**Requirement Title**	**CSC Name**	**CSU Name**
[A-2]	5.1.0.0.4 Signal Data Segment Duration 5.1.0.0.5 Data Playback Time 5.1.1.1.3 Sampling Rate 5.1.2.0.2 Data Segment Length 5.3.0.0.1 Signal Playback	SDSP	SDIO

Table 9-4 DSP/Testbed Interface Software Requirements Traceabiltiy

Spec #	Requirement Title	CSC Name	CSU Name
[A-1]	3.2.2.1.1 Concurrency	SMC DIF	CSIT CDSI
[A-1]	3.2.2.1.2 Sources and Sinks	SMC DIF	CSIT CDSI
[A-1]	3.2.2.1.3 Application Protocol	SMC DIF	CSIT CDSI
[A-1]	3.2.2.1.4 Preconditions	SMC DIF	CSIT CDSI
[A-1]	3.2.2.2.1 Detection Data Element	DIF DMT	CDSI CSIP
[A-1]	3.2.2.2.2 Intermediate Data Element	DIF DMT	CDSI CSIP
[A-1]	3.2.3.1.1 Concurrency	SMC DIF	CSIT CDSI
[A-1]	3.2.3.1.2 Sources and Sinks	SMC DIF	CSIT CDSI
[A-1]	3.2.3.1.3 Application Protocol	SMC DIF	CSIT CDSI
[A-1]	3.2.3.2.1 DSP Commands	SMC DIF	CSIT CDSI

Table 9-5 Testbed Hardware Requirements Traceabillity

Spec #	Requirement Title	HWCI Name	Major Component Name
[A-2]	3.4.2.0.1 Video Display	Cont, RSS	Video Display
[A-2]	3.4.2.0.2 Keyboard	Cont, RSS	Keyboard
[A-2]	3.4.2.0.3 Pointing Device	Cont, RSS	Mouse
[A-2]	3.4.2.0.4 Printer	Cont	Printer
[A-2]	3.4.2.0.5 Program Load/Store Device	Cont,	150 MB Tape Drive
		Cont, RSS	CD Drive
[A-2]	3.4.2.0.6 Archive Device	Cont, RSS	5 GB Tape Drive
[A-2]	3.4.3.2.1 DSP Hardware Interface	Cont, RSS	SPARCstation 10
		RTSG	IOP
		RTSG	IOP Adaptor
		RTSG	Backplane
		RTSG	Chassis
[A-2]	3.4.4.1.1 Alternating Current Power	Cont, RSS	SPARCstation 10
		Cont, RSS	Video Display
		Cont	150 MB Tape Drive
		Cont, RSS	5 GB Tape Drive
		Cont, RSS	CD Drive
		Cont, RSS	Disk
		Cont	Printer
		RTSG	Power Supply
		RTSG	Chassis
[A-2]	3.4.4.2.1 Cables	Cont, RSS	SPARCstation 10
		Cont, RSS	Video Display
		Cont	150 MB Tape Drive
		Cont, RSS	5 GB Tape Drive
		Cont, RSS	CD Drive
		Cont, RSS	Disk
		Cont	Printer
		RTSG	Backplane
		RTSG	IOP
		RTSG	Chassis

Spec #	Requirement Title	HWCI Name	Major Component Name
[A-2]	3.4.4.2.2 Connectors	Cont, RSS	SPARCstation 10
		Cont, RSS	Video Display
		Cont	150 MB Tape Drive
		Cont, RSS	5 GB Tape Drive
		Cont, RSS	CD Drive
		Cont, RSS	Disk
		Cont	Printer
		RTSG	Backplane
		RTSG	IOP
		RTSG	Chassis
[A-2]	3.4.4.3.1 Power Source	Cont, RSS	SPARCstation 10
		Cont, RSS	Video Display
		Cont	150 MB Tape Drive
		Cont, RSS	5 GB Tape Drive
		Cont, RSS	CD Drive
		Cont, RSS	Disk
		Cont	Printer
		RTSG	Power Supply
		RTSG	Chassis
[A-2]	3.5.1.2.1 Graphics Controller	Cont, RSS	SPARCstation 10
[A-2]	3.5.1.2.3 Real time output	RTSG	IOP
[A-2]	3.5.1.2.4 Non-volatile On-line storage	Cont, RSS	SPARCstation 10
		Cont, RSS	Disk
[A-2]	3.5.1.2.5 High speed memory capacity	RTSG	TPM

Note: Cont. is Controller Workstation; RSS is RSS Workstation; RTSG is Real-Time Signal Generator.

Table 9-6 Controller Hardware Requirements Traceability

Spec #	Requirement Title	HWCI Name	Major Component Name
[A-2]	4.1.2.1.1 Multitasking Environment	Cont.	SPARCstation 10
[A-2]	4.1.2.2.2 Interface Devices	Cont.	SPARCstation 10
[A-2]	4.1.2.3.2 Initialization	Cont. Cont.	SPARCstation 10 Keyboard
[A-2]	4.1.2.4.1 Floating-Point Computation	Cont.	SPARCstation 10
[A-2]	4.1.2.4.2 Integer Computation	Cont.	SPARCstation 10
[A-2]	4.2.1.2.1 Controller/HMI Hardware Interface	Cont. Cont. Cont. Cont. Cont. Cont. Cont. Cont.	SPARCstation 10 Keyboard Video Display Mouse 150 MB Tape Drive 5 GB Tape Drive CD Drive Printer
[A-2]	4.2.3.0.1 Controller/DSP Interface	Cont.	SPARCstation 10
[A-2]	4.3.2.0.1 HMI Performance	Cont. Cont. Cont. Cont. Cont. Cont. Cont.	Keyboard Video Display Mouse 150 MB Tape Drive 5 GB Tape Drive CD Drive Printer
[A-2]	4.3.2.0.2 Non-volatile On-line Storage	Cont. Cont.	SPARCstation 10 Disk
[A-2]	4.3.2.0.3 Graphics Controller	Cont.	SPARCstation 10

Note: Cont. is Controller Workstation

Table 9-7 RSS Hardware Requirements Traceability

Spec #	Requirement Title	HWCI Name	Major Component Name
[A-2]	5.2.1.2.1 Controller/RSS Hardware Interface	RSS RTSG	Ethernet Executive Controller
[A-2]	5.2.2.2.1 ESP/RSS Hardware Interface	RTSG RTSG RTSG RTSG	Backplane IOP Chassis IOP Adapter
[A-2]	5.3.0.0.1 Signal Playback	RTSG RSS	IOPs SPARCstation 10
[A-2]	5.3.0.0.2 Signal Generation	RSS RSS	Vector Coprocessor SPARCstation 10

Note: RSS is RSS Workstation; RTSG is Real-Time Signal Generator

Table 9-8 DSP/Testbed Interface Hardware Requirements Traceability

Spec #	Requirement Title	HWCI Name	Major Component Name
[A-1]	3.3.0.0.1 Allocation to Physical Links Interface	Cont, RSS RTSG RTSG	SPARCstation 10 Executive Controller Backplane
[A-1]	3.3.0.0.2 Physical Connection	Cont, RSS RTSG RTSG	SPARCstation 10 Executive Controller Backplane
[A-1]	3.3.2.1.1 General	Cont, RSS RTSG RTSG	SPARCstation 10 Executive Controller Backplane
[A-1]	3.3.2.1.2 Physical Link Control	Cont, RSS RTSG RTSG	SPARCstation 10 Executive Controller Backplane
[A-1]	3.3.2.2.1 Cables and Connectors	Cont,RSS RTSG RTSG	SPARCstation 10 Executive Controller Backplane
[A-1]	3.3.2.3.1 Signal Levels	Cont, RSS RTSG RTSG	SPARCstation 10 Executive Controller Backplane

Note: Cont. is Controller Workstation; RSS is RSS Workstation; RTSG is Real-Time Signal Generator.

SECTION TEN

NOTES

10.1 ACRONYMS AND ABBREVIATIONS

A	Ampere
AC	Alternating Current
A/D	Analogue-to-Digital
ADS	Application Development System
AES	Application Environmental Specification
ANSI	American National Standards Institute
ASAR	Arrested Synthetic Aperture Radar
AWG	American Wire Gauge
CC	Clutter Cancellation
CD	Compact Disk
CD-ROM	Compact Disk Read Only Memory
CDADL	C-Based Design and Documentation Language
CDAT	Controller: Data Analysis (CSU)
CDSH	Controller: Simulation Run HMI (CSU)
CDSI	Controller: DSP Simulation Interface (CSU)
CEND	Controller: Environment Definition (CSU)
CENH	Controller: Environment HMI (CSU)
CERR	Controller: Error Logging (CSU)
CFA	Corporate-Fed Array
CFM	Cubic Feet per Minute
CGEH	Controller: Control Generation HMI (CSU)
CGEP	Controller: Signal Generation Products (CSU)
CGET	Controller: Generation Server (CSU)
CLCC	Ceramic Leaded Chip Carrier
CLR	Controller (functional area)
CML	Configuration Management Library
CMT	Control and Management (CSC)
COTS	Commercial Off-the-Shelf
CP	Coherent Processing
CPI	Coherent Processing Interval
CPU	Central Processing Unit
CRAD	Controller: Satellite/Radar Definition (CSU)
CRDH	Controller: Satellite/Radar HMI (CSU)

CRGI	Controller: RSS Generation Interface (CSU)
CRPI	Controller: RSS Playback Interface (CSU)
CSA	Canadian Standards Association
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSDH	Controller: Simulation Run HMI (CSU)
CSID	Controller: Simulation Run Definition (CSU)
CSIH	Controller: Control Simulation HMI (CSU)
CSIP	Controller: Simulation Products (CSU)
CSIT	Controller: Simulation Server (CSU)
CSPI	CSPI, Inc. (manufacturer of co-processor)
CSU	Computer Software Unit
CSYS	Controller: System Services
CTI	Contiguous Time Interval
CUIS	Controller: User Interface Services
CVAL	Controller: Data Validation (CSU)
DAN	Data Analysis (CSC)
DARTS	Design Approach for Real Time Systems
DC	Direct Current
DI	Dwell Interval
DIF	Controller/DSP Interface (CSC)
DMT	Data Management (CSC)
DPCA	Displaced Phase Centre Antenna
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
DSPI	Digital Signal Processor Interface
EC	Executive Controller
ECCM	Electronic Counter-counter Measures
EMI	Electromagnetic Interference
EPROM	Erasable Programmable Read-Only Memory
ESP	ECCM Signal Processor
FA	Frequency Alignment
FCC	Federal Communication Commission
FPGA	Field Programmable Gate Array
GB	Gigabyte
GND	Ground
GSP	General Signal Processor
GUI	Graphical User Interface
HMI	Human Machine Interface
HWCI	Hardware Configuration Item
I	In-Phase
IEEE	Institute of Electrical and Electronics Engineers
I/F	Interface
IOP	Input/Output Processors
IPC	Interprocessor Communication
ISDN	Integrated Services Digital Network
ISSPL	Industry-Standard Signal Processing Library

kg	kilogram
LAN	Local Area Network
mm	Millimetre
MASR	Multiple Antenna Surveillance Radar
MByte	Megabyte
MFLOPS	Million Floating-Point Operations per Second
MTBF	Mean Time Between Failure
NCI	Non-Coherent Interval (or Integration)
NFS	Network File System
OnCE	On-Chip Emulation
PC	Pulse Compression
PD	Pulse Doppler
PDL	Program Design Language
PE	Processing Element
PLCC	Plastic Leaded Chip Carrier
PMC	Platform Motion Compensation
Q	Quadrature
RCS	Radar Cross Section
RFI	Radio Frequency Interference
RGI	RSS Generation
RIF	Controller/RSS Interface (CSC)
RPC	Remote Procedure Call
RPI	RSS Playback
RRA	Rotating Reflector Antenna
RSP	Radar Signal Processor
RSS	Radar Signal Simulator
RTSG	Real-Time Signal Generator
SBIT	(RSS) Burst Initialization (CSC)
SBMT	(RSS) Burst Modelling (CSC)
SBR	Space Based Radar
SC	Sampling Correction
SCSI	Small Computer Systems Interface
SGD	Signal Generation Definition (CSC)
SGEN	Signal Generation Controller (CSC)
SMC	System Monitor/Control (CSC)
SPB	Synchronous Parallel Bus
SPD	Signal Processing Demonstrator
SPMT	(RSS) Playback Manager (CSC)
SRD	Simulation Run Definition (CSC)
SRUN	(RSS) Run Initialization (CSC)
SRSD	Simulated Radar Signal Data
SSGM	Signal Generation Manager (CSC)
SSP	Sub-Satellite Point
SUTL	(RSS) Utilities (CSC)
TDIT	(RTSG) DSP Interface (CSC)
TIOP	(RTSG) DSP Output (CSC)

TPM	Two-Part Memory
TRSV	(RTSG) RPC Server (CSC)
UIS	User Interface Services
UL	Underwriters Laboratories, Inc.
UTL	Utilities (CSC)
VASP	Versatile Array Signal Processor
VME	Versa Module European
XDR	eXternal Data Representation

APPENDIX A

CONTROLLER CSC DESIGN PROCESS

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

Page

APPENDIX A CONTROLLER CSC DESIGN PROCESS A-1

A1 Design Models A-1
A2 Processor Model A-1
A3 Tasking Model A-1
A4 Package Model A-4

A4.1 Package Methodology A-4
A4.2 Package Standards and Conventions A-6
A4.3 Package Decomposition A-6

A5 Data File Subsystem A-7

A5.1 Operational Environment A-7
A5.2 Operational Data Files A-7

A6 Task Implementation A-23

A6.1 Satellite/Radar Definition Task A-23
A6.2 Environment Definition Task A-27
A6.3 Simulation Run Definition Task A-30
A6.4 Generation Server Task A-36
A6.5 Monitor/Control Simulation Task A-38
A6.6 Simulation Server Task A-42
A6.7 Data Analysis Task A-45

A7 Error Handling A-46

A7.1 Objectives A-46
A7.2 Definitions A-46
A7.3 Propagation A-47
A7.4 Conventions A-47
A7.5 Logging A-47
A7.6 CERR Package A-48
A7.7 Error Log Files A-48

FIGURES

Page

A-1 CONTROLLER TASKING MODEL DIAGRAM A-3
A-2 CONTROLLER PACKAGE MODEL DIAGRAM A-5
A-3 CDRH PACKAGE STRUCTURE CHART A-26
A-4 ENVIRONMENT DEFINITION HMI TOP LEVEL CONTROL FLOW A-28
A-5 DISCRETE CLUTTER INTERFACE HANDLING A-28
A-6 CSDH PACKAGE STRUCTURE CHART A-34
A-7 CGEH PACKAGE STRUCTURE CHART A-37
A-8 CGET PACKAGE STRUCTURE CHART A-39
A-9 CSIH PACKAGE STRUCTURE CHART A-41
A-10 CSIT PACKAGE STRUCTURE CHART A-44

TABLES

A-1 PACKAGE CROSS REFERENCE TABLE A-6
A-2 PARAMETER GROUP DATA FILES A-10
A-3 RADAR SIGNAL GROUP DATA FILES A-13
A-4 PRODUCT GROUP DATA FILES A-14
A-5 LOG GROUP DATA FILES A-22
A-6 CENH CSU TESTING MATRIX A-29

APPENDIX A

CONTROLLER CSC DESIGN PROCESS

A1 Design Models

Section B1 of the Testbed preliminary design document [R-3] provided a description of the implementation design models used in the design of the Testbed Controller. The implementation models described included the following:

a) Processor Model - defines the processors required to carry out the functions allocated from the essential model;

b) Task Model - defines the tasks (or chunks of software) that can be scheduled as a unit - identifies the concurrency within each processor identified in the processor model;

c) Package Model - defines the software organization (or packaging).

The following sections identify any changes to the above implementation models as described in the preliminary design document. Much more detailed information will be provided for the package model, decomposing the CSCs identified in the preliminary design into CSUs.

A2 Processor Model

There are no changes to the processor model as described in the preliminary design document [R-3]. The Controller Workstation and RSS Workstation can perform all the functionality associated with the Controller CSC.

A3 Tasking Model

Changes have been made to the tasking model as described in the preliminary design document [R-3]. The changes can best be summarized in the following manner:

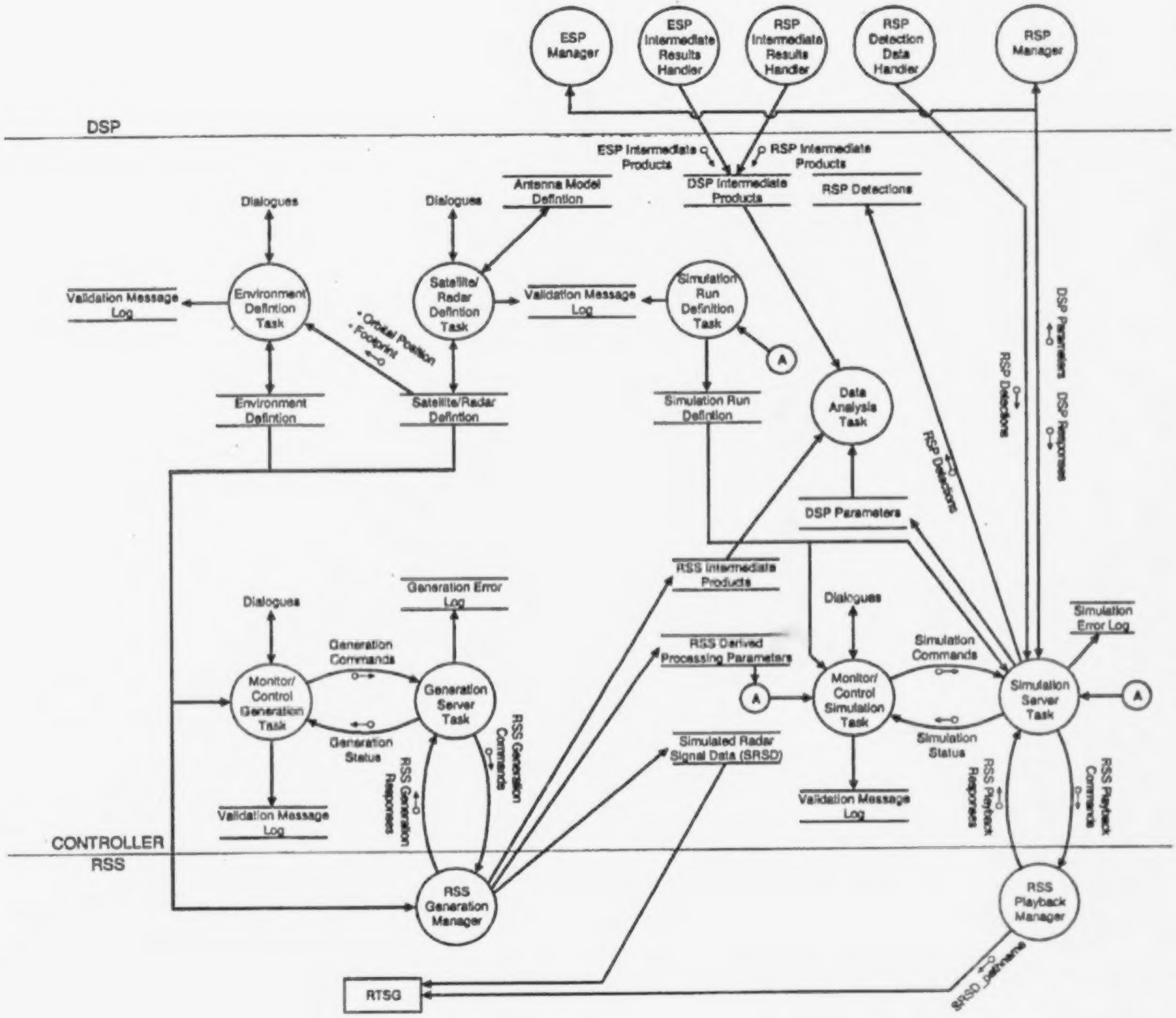
a) Eliminate need for overall SPD session control HMI and and treating the definition of satellite/radar, environment and simulation run parameters as three separate HMI activities. This resulted in decomposing the Radar Definition task into three seperate tasks, each dealing with one of the above three parameter definitions. This also eliminated need for overall session errorlog;

b) Removed requirement for batching or queuing of signal generation and simulation run activities. This resulted in eliminating activity status files and in changes to the role of the generation and simulation server tasks (no longer need to maintain internal queue of activities);

c) Removed requirement for step mode capability during a simulation thus changing the content of the playback parameters sent by the Controller Simulation Server task and the RSS Playback task;

d) Eliminate the automatic creation and deletion of the Generation Server and Simulation Server tasks in between generation or simulation activities. These server tasks will now be considered as daemons and will remain in existance until explicitly shutdown by the user.

Figure A-1 contains the revised tasking model diagram reflecting the above changes. The diagram identifies the individual tasks and the interactions and data transfers which take place between tasks and between shared data stores.


ESP Manager
ESP Intermediate Results Handler
RSP Intermediate Results Handler
RSP Detection Data Handler
RSP Manager
DSP
ESP Intermediate Products
RSP Intermediate Products
DSP Intermediate Products
RSP Detections
Antenna Model Defintion
Dialogues
Dialogues
Validation Message Log
Environment Defintion Task
Satellite/ Radar Defintion Task
Validation Message Log
Simulation Run Definition Task
A
• Orbital Position
• Footprint
Environment Defintion
Satellite/Radar Defintion
Simulation Run Defintion
Data Analysis Task
DSP Parameters
DSP Responses
RSP Detections
RSP Detections
DSP Parameters
RSS Intermediate Products
Generation Error Log
Dialogues
Dialogues
Simulation Error Log
RSS Derived Processing Parameters
Generation Commands
Monitor/ Control Generation Task
Generation Server Task
Generation Status
RSS Generation Commands
RSS Generation Responses
A
Monitor/ Control Simulation Task
Simulation Commands
Simulation Server Task
A
Simulation Status
Simulated Radar Signal Data (SRSD)
Validation Message Log
Validation Message Log
RSS Playback Responses
RSS Playback Commands
CONTROLLER
RSS
RSS Generation Manager
RSS Playback Manager
RTSG
SRSD_pathname

AI-7130-D2R2\CW

FIGURE A-1 CONTROLLER TASKING MODEL DIAGRAM

A4 Package Model

As described in the Testbed preliminary design document [R-3], the package model used within this design will use the Computer Software Component (CSC) and Computer Software Unit (CSU) concept of defining the software organization.

The preliminary design document [R-3] identified the software packages to the CSC level while this detailed design document decomposes the software packages to the CSU level. In this document, the term software package is synonymous with CSU. Figure A–2 contains the package model diagram with the associated CSU names.

Changes have been made to the CSC package model as described in the preliminary design. These changes are described in each CSC description provided under section 3.2 of this document and will not be repeated in this section.

A4.1 Package Methodology

A software package is usually used to define a functional component of the system design. It is defined in terms of what it looks like to the other functional components. Packages are created with two design constructs; the package specification and the package body.

The package specification provides an external view of the package. It is implemented in the form of an interface file completely defining the external interface to the package, including:

a) data types that are accessible outside the package;

b) services that are accessible outside the package (function names and parameter declarations only).

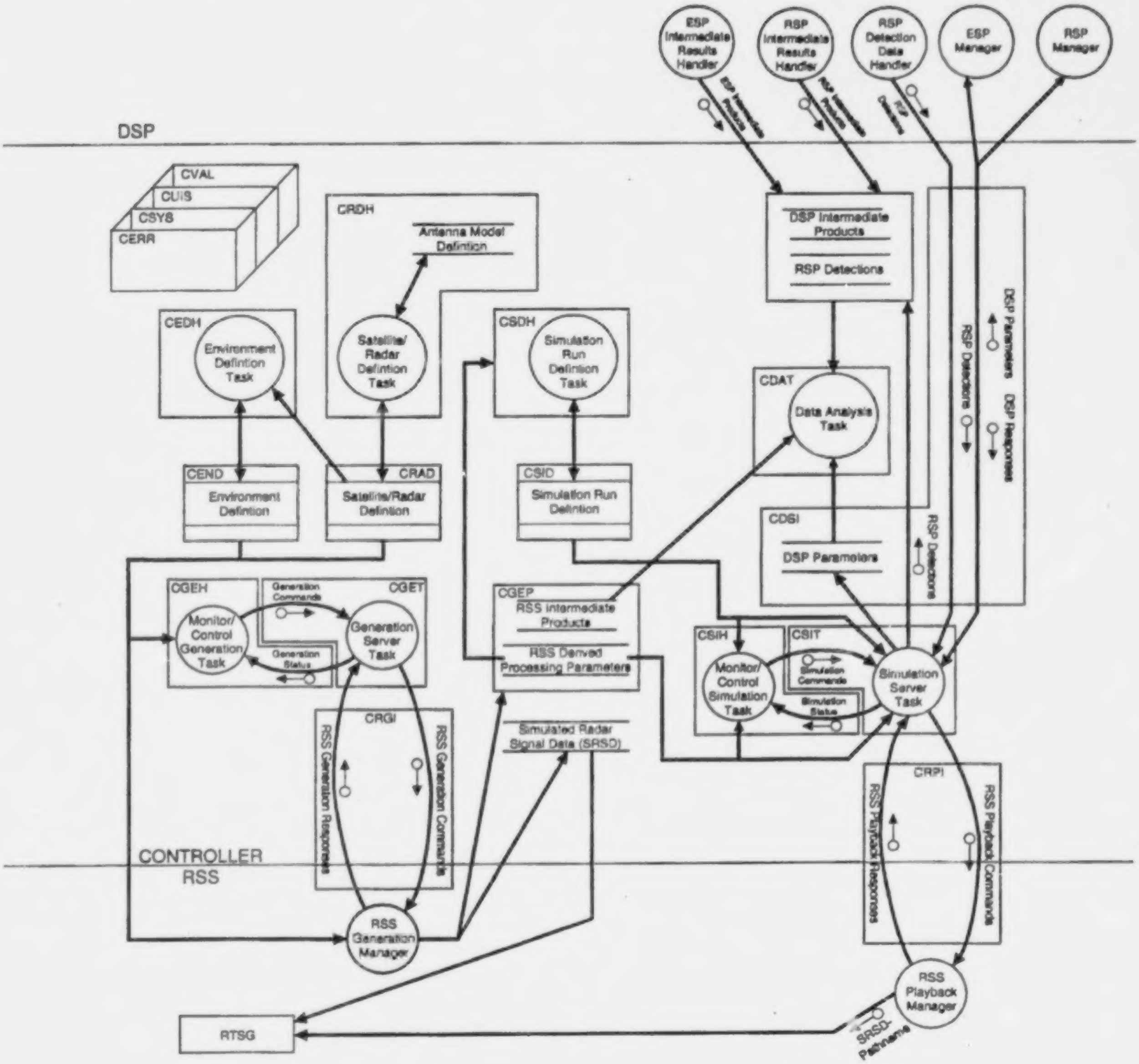
Package users are restricted to using the exported data types and services specified in this interface.

The package body is where we can hide information from the user and consists of the following:

a) data types and services that are only accessible from within the package;

b) coded functions that have been declared in the interface file;

c) local and private coded functions required internally to support the package.


ESP Intermediate Results Handler
RSP Intermediate Results Handler
RSP Detection Data Handler
ESP Manager
RSP Manager
ESP Intermediate Products
RSP Intermediate Products
RSP Detections
DSP
CVAL
CUIS
CSYS
CERR
CRDH
Antenna Model Definition
DSP Intermediate Products
RSP Detections
CEDH
Environment Definition Task
Satellite/ Radar Definition Task
CSDH
Simulation Run Definition Task
DSP Parameters
RSP Detections
DSP Responses
CDAT
Data Analysis Task
CEND
Environment Definition
CRAD
Satellite/Radar Definition
CSID
Simulation Run Definition
CDSI
DSP Parameters
RSP Detections
CGEH
Generation Commands
CGET
Monitor/ Control Generation Task
Generation Server Task
Generation Status
CGEP
RSS Intermediate Products
RSS Derived Processing Parameters
CSIH
CSIT
Monitor/ Control Simulation Task
Simulation Commands
Simulation Status
Simulation Server Task
CRGI
RSS Generation Responses
RSS Generation Commands
Simulated Radar Signal Data (SRSD)
CRPI
RSS Playback Responses
RSS Playback Commands
CONTROLLER
RSS
RSS Generation Manager
RSS Playback Manager
RTSG
SRSD-Pathname

AI-7130-D1R2/CW

FIGURE A-2 CONTROLLER PACKAGE MODEL DIAGRAM

A4.2 Package Standards and Conventions

Package specifications will be provided in terms of include files (C language header files). Each include file will contain syntactically correct C code consisting of data declarations and of function declarations with proper parameter definitions. For include file format and naming conventions for files and identifiers, refer to section on detailed design standards and conventions in Appendix C.

Package bodies will be provided in terms of source files (C language .c files). Each source file will contain the PDL logic describing the algorithms implemented by the modules within the package. The subsequent implementation phase will use these source files and merely convert the PDL logic into compilable C software. For source file format and naming conventions for files and identifiers, refer to section on PDL file standards and conventions in Appendix C. Appendix C also provides a description of the PDL guidelines, including PDL syntax, that were followed in generating the PDL logic.

A4.3 Package Decomposition

The packages identified are listed in the package cross-reference Table A-1. This provides a project unique identifier and a reference to appropriate sections in the Controller PDL (Appendix D), containing the interface specification and PDL logic information.

TABLE A-1 PACKAGE CROSS REFERENCE TABLE

Package (CSU) Name and Project Unique Identifier	Appendix D Section
Satellite/Radar HMI (CRDH)	D1.1
Environment HMI (CENH)	D1.2
Simulation Run HMI (CSDH)	D1.3
RSS Generation Interface (CRGI)	D1.4
RSS Playback Interface (CRPI)	D1.5
DSP Simulation Interface (CDSI)	D1.6
Data Analysis (CDAT)	D1.7
Satellite/Radar Definition (CRAD)	D1.8
Environment Definition (CEND)	D1.9
Simulation Run Definition (CSID)	D1.10
Simulation Products (CSIP)	D1.11
Signal Generation Products (CGEP)	D1.12
Control Generation HMI (CGEH)	D1.13
Generation Server (CGET)	D1.14
Control Simulation HMI (CSIH)	D1.15
Simulation Server (CSIT)	D1.16
Data Validation (CVAL)	D1.17
Error Logging (CERR)	D1.18
System Services (CSYS)	D1.19
User Interface Services (CUIS)	D1.20

A5 Data File Subsystem

This section defines the data file subsystem of the Testbed Controller for the operational SPD system. This includes defining the naming conventions for the user entered or system generated data files and the file subsystem structure indicating how the files will be organized.

A5.1 Operational Environment

The deliverable SPD will consist of the RSS and Controller workstation. Each workstation will have two 424 MB local disks. The local disks will consist of system files, COTS software and SPD developed application software, including source. This will consume most of the local disks.

The Controller workstation will also include one 1.3 GB disk and the RSS workstation will include two 1.3 GB disks. These disks will be NFS mounted and accessible from each workstation. Refer to the equipment block diagrams, Figures 3-6 and 3-8 in the Testbed preliminary design document [R-3], for the configuration of the disk drives within the RSS and Controller workstations. The data file subsystem as described in the following sections, will be distributed accross the three 1.3 GB disks.

A5.2 Operational Data Files

The "operational" data files for the Testbed Controller will consist of any files generated as result of user entering parameter data through the SPD HMI, data files generated by signal generation and simulation activities and analysis files generated from data analysis activities. These data files will be defined by the following file groups:

a) parameter files
b) radar signal (SRSD) files
c) product files
d) analysis files
e) log files

A5.2.1 File Organization

For each of the five file groups, a seperate sub-directory will be provided to store the associated file(s). These sub-directories will be organized directly under the root directory for the "user-id" and shown as follows:

```
/"user-id"
            /parm
                                        .
                                        .
                                        .
            /srsd
                                        .
                                        .
                                        .
            /prod
                                        .
```

•
•

/anal

•
•
•

/logs

•
•
•

The following Unix system environment variables will be defined for each "user-id" within each user account to define the pathname to the corresponding sub-directory (a generic script file will be provided which can be customized by the individual user, i.e., replacing the generic "user-id" in the script file with the actual user name):

- PARM_DIR
- SRSD_DIR
- PROD_DIR
- ANAL_DIR
- LOGS_DIR

The Testbed Controller software will use these environment variables as the default pathname for accessing the appropriate files.

A5.2.2 File Allocation

The allocation of file groups to physical disks will be provided via Unix symbolic link commands. This will allow for easy configuration (on a per user-id basis) of where the files are actually to reside.

The current allocation will have the Simulated Radar Signal Data (SRSD) files allocated entirely to one of the 1.3 GB disks on the RSS workstation. The total volume of signal data stored on-line is bounded by the 1.3 GB capacity of the disk. Given that each file is limited to the playback memory maximum size of 192 MB, at least 5 SRSD files may be stored on-line at once. The size of typical SRSD files is likely to be less than the maximum, allowing more files to be stored. The other four file groups will be allocated to the other 1.3 GB disk on the RSS workstation and the 1.3 GB disk on the Controller workstation.

A5.2.3 File Characteristics

The following sections describe the file characteristics for each of the files within the five file groups as identified in A5.2 . The file characteristics include naming convention, pathname of sub-directory, description of file content, and maximum file size.

The naming conventions provide a standard approach to accessing the various files generated by the SPD. The convention is based on the following naming scheme:

file name = <id>.<category>.<type>

where:

<id>: this is a 1-12 alphanumeric identifier speficied by the user via HMI dialogues;

<category>: this identifies the category (or sub-category) of the data file and is provided by the system. The following are the available categories:

pa:	parameter group of data files
sg:	radar signal data group of files
ir:	intermediate results (product) group of files
dt:	detection results (product) group of files
lg:	error log group of files

Note: There is no specified category identifier associated with analysis (PV-Wave) group of data files as there should be no restriction as to what file names the user may want to use in generating analysis results from the PV-Wave environment.

<type>: this identifies the type of data file within the specified data file category and is provided by the system. Refer to each category for specific file types.

Tables A-2 through A-5 define the file characteristics for each type of data file within the specified file categories. The maximum file size indicates an upper bound only for an intermediate result, and does not represent typical sizes. Most intermediate results will be smaller owing to smaller signal data files, and data reduction/decimation that may be applied to the intermediate data before storage. Typical intermediate files are expected to range in the 1-10 MB size.

Table A-2 Parameter Group Data Files (Page 1 of 3)

File Name	Root Directory	Description	Structure Definition	Max Size (MB)
"id".pa.sar_def	PARM_DIR	Satellite and radar parameters entered by user via HMI dialogues for Satellite/Radar Definition. "id" specified by user when saving file.	t_crad_sat_def	0.3
"id".pa.env_def	PARM_DIR	Environment parameters entered by user via HMI dialogues for Environment Definition. "id" Environment Definition identifier, specified by user when saving file.	t_cend_env	1.0
"id".pa.sim_def	PARM_DIR	Simulation run parameters entered by user via HMI dialogues for Simulation Run Definition. "id" Simulation Run Definition identifier, specified by user when saving file.	t_csid_sim_def	0.1
"id".pa.rss_prm	PARM_DIR	RSS derived parameters generated by the RSS during a signal generation activity. "id" generation run identifier, specified when starting signal generation.	t_cgep_RSS_process_p arms	0.8

Table A-2 Parameter Group Data Files (Page 2 of 3)

"id".pa.dsp_prm	PARM_DIR	DSP processing parameters (sent to the ESP and RSP) - generated by the Testbed Controller during a simulation run activity. "id" simulation run identifier, specified when starting simulation run.	t_cdsi_DSP_process_ parms	1
File Name	Root Directory	Description	Structure Definition	Max Size (MB)
"id".pa. sim__def.pa.sar_def	PARM_DIR	Satellite and radar parameters associated with a Simulaton Run Definiton - copied by the Testbed Controller during a Simulation Run Definition activity. "id" Simulation Run identifier, the same as the associated Simulation Run Definition file.	t_crad_sat_def	0.3
"id".pa. rss_prm.pa.sar_def	PROD_DIR	Satellite and radar parameters associated with a signal generation - copied by the Testbed Controller during a signal generation activity. "id" generation run identifier, the same as the associated RSS derived processing parameters file.	t_crad_sat_def	0.3
"id".pa.ant_RRA or "id".pa.ant_CFA	PARM_DIR	Re-usable CFA or RRA antenna model parameters. Generated by user during entry of satellite/radar parameter definition. "id" antenna model identifier, specified when defining satellite/radar definition.	t_crad_RRA_file or t_crad_CFA_file	0.02

Table A-2 Parameter Group Data Files (Page 3 of 3)

“id”.pa.if_equal	PARM_DIR	Stored file of I/F and equalizer characteristics. Refer to Satellite/Radar HMI parameter table (section 2.5.5.3). “id” file identifier of stored file, selected when entering receiver front end parameters.	t_crad_IF_equalization	0.01
File Name	Root Directory	Description	Structure Definition	Max Size (MB)
“id”.pa.ant_patrn	PARM_DIR	Stored file of antenna element pattern model for Tx and Rx antenae. Refer to Satellite/Radar HMI parameter table (section 2.2.10.1 and 2.3.17.1). “id” file identifier of stored file, selected when defining antenna model (CFA or RRA).	ASCII file in Phasar-C format	1.0

Table A-3 Radar Signal Group Data Files

File Format	Root Directory	Description	Structure Definition	Max Size (MB)
"id".sg.srsd	SRSD_DIR	Simulated Radar Signal Data generated by the RSS during a signal generation activity. Max size based on RTSG memory capacity. "id" generation run identifier, specified when starting signal generation.	t_sgen_srsd_header t_sgen_cpi_data	192

Table A-4 Product Group Data Files (Page 1 of 8)

File Format	Root Directory	Description	Structure Definition	Max Size (MB)
"id".ir.rss_clut	PROD_DIR	Clutter signal data (excludes affects of external noise, jammers and targets). Product file generated by the RSS and consists of data for all antenna(s). "id" generation run identifier, specified when starting signal generation.	t_cgep_header_block t_cgep_control_block signal data	384
"id".ir.rss_exn	PROD_DIR	External (galactic) noise signal data. Product file generated by the RSS and consists of data for all antenna(s). "id" generation run identifier, specified when starting signal generation.	t_cgep_header_block t_cgep_control_block signal data	384
"id".ir.rss_jam	PROD_DIR	Signal data for the selected jammers. Product file generated by the RSS and consists of data for all antenna(s). "id" generation run identifier, specified when starting signal generation.	t_cgep_header_block t_cgep_control_block signal data	384
"id".ir.rss_tar	PROD_DIR	Signal data for the selected targets. Product file generated by the RSS and consists of data for all antenna(s). "id" generation run identifier, specified when starting signal generation.	t_cgep_header_block t_cgep_control_block signal data	384

Table A-4 Product Group Data Files (Page 2 of 8)

File Format	Root Directory	Description	Structure Definition	Max Size (MB)
"id".ir.rss_comp	PROD_DIR	Composite signal data (includes external noise, clutter, targets and jammers) received at the antenna front end. Product file generated by the RSS and consists of data for all antenna(s). "id" generation run identifier, specified when starting signal generation.	t_cgep_header_block t_cgep_control_block signal data	384
"id".ir.rss_post_nl	PROD_DIR	Composite signal data after the application of effects of non-linearity. Product file generated by the RSS and consists of data for all antenna(s). "id" generation run identifier, specified when starting signal generation.	t_cgep_header_block t_cgep_control_block signal data	384
"id".ir.rss_if_filt	PROD_DIR	Composite signal data after application of the effects of I/F filtering. Product file generated by the RSS and consists of data for all antenna(s). "id" generation run identifier, specified when starting signal generation.	t_cgep_header_block t_cgep_control_block signal data	384
"id".ir.rss_post_sd	PROD_DIR	Signal data after application of the affects of synchronous detection. Product file generated by the RSS and consists of data for all antenna(s). "id" generation run identifier, specified when starting signal generation.	t_cgep_header_block t_cgep_control_block signal data	384

Table A-4 Product Group Data Files (Page 3 of 8)

File Format	Root Directory	Description	Structure Definition	Max Size (MB)
"id".ir.rss_ant_patrn	PROD_DIR	Antenna pattern product file generated by the RSS and consists of data for all antenna(s). "id" generation run identifier, specified when starting signal generation.	t_cgep_antenna	2.0
"id".ir.rss_filt_rsp	PROD_DIR	I/F filter response product file generated by the RSS and consists of data for all antenna(s). "id" generation run identifier, specified when starting signal generation.	t_cgep_if_filter	2.0
"id".ir.rss_post_ad	PROD_DIR	Signal data after the application of the affects of A/D conversion. Product file generated by the RSS and consists of data for all antenna(s). "id" generation run identifier, specified when starting sgnal generation.	t_cgep_header_block t_cgep_control_block signal data	384
"id".ir.esp_inp	PROD_DIR	Time domain radar signal data received from the Testbed. Product file generated by the ESP. "id" simulation run identifier, specified when starting simulation run.	t_cgep_header_block t_cgep_control_block signal data	384

Table A-4 Product Group Data Files (Page 4 of 8)

File Format	Root Directory	Description	Structure Definition	Max Size (MB)
"id".ir.esp_post_sc	PROD_DIR	Time domain bursts after sampling correction. Product file generated by the ESP. "id" simulation run identifier, specified when starting simulation run.	t_cgep_header_block t_cgep_control_block signal data	384
"id".ir.esp_post_an	PROD_DIR	Time domain bursts, for one channel, after adaptive nulling. Product file generated by the ESP. "id" simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	384
"id".ir.esp_post_pdb	PROD_DIR	Time domain bursts, for one channel, after pre-doppler blanking. Product file generated by the ESP. "id" simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	384
"id".ir.esp_post_masr	PROD_DIR	Time domain bursts after MASR DPCA. Product file generated by the ESP. "id" simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	384
"id".ir.esp_post_fa	PROD_DIR	Time domain bursts after frequency alignment. Product file generated by the ESP. "id" simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	384

Table A-4 Product Group Data Files (Page 5 of 8)

File Format	Root Directory	Description	Structure Definition	Max Size (MB)
“id”.ir.esp_post_pd	PROD_DIR	Bursts after pulse doppler forward FFT. Product file generated by the ESP. “id” simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	384
“id”.ir.esp_nwt_vec	PROD_DIR	Adaptive nulling weight vectors. Product file generated by the ESP. “id” simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	1.0
“id”.ir.esp_corr_mx	PROD_DIR	Correlation matrices; R-matrices and r-vectors. Product file generated by the ESP. “id” simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	2.1
“id”.ir.esp_csky_fact	PROD_DIR	Cholesky factor matrices; U-matrices and z-vectors. Product file generated by the ESP. “id” simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	17
“id”.ir.esp_an_snap	PROD_DIR	Adaptive nulling snapshots. Product file generated by the ESP. “id” simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	384

Table A-4 Product Group Data Files (Page 6 of 8)

File Format	Root Directory	Description	Structure Definition	Max Size (MB)
“id”.ir.esp_pd_win	PROD_DIR	Pulse doppler window function. Product file generated by the ESP. “id” simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	0.4
“id”ir.rsp_inp	PROD_DIR	Range-doppler bursts as input to RSP. Product file generated by the RSP. “id” simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	768
“id”.ir.rsp_post_pc	PROD_DIR	Range-doppler bursts after pulse compression. Product file generated by the RSP. “id” simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	768
“id”.ir.rsp_post_cc	PROD_DIR	Range-doppler bursts after ASAR DPCA clutter cancellation. Product file generated by the RSP. “id” simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	768
“id”.ir.rsp_post_rcs	PROD_DIR	Range-doppler bursts after magnitude-squared detection. Product file generated by the RSP. “id” simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	768

Table A-4 Product Group Data Files (Page 7 of 8)

File Format	Root Directory	Description	Structure Definition	Max Size (MB)
"id".ir.rsp_post_rdb	PROD_DIR	Range-doppler bursts after range-doppler blanking. Product file generated by the RSP. "id" simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	768
"id".ir.rsp_post_nci	PROD_DIR	Integrated power matrix, and integrated monopulse angle. Product file generated by the RSP. "id" simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	768
"id".ir.rsp_mono_mx	PROD_DIR	Monopulse angle matrix before averaging. Product file generated by the RSP. "id" simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	768
"id".ir.rsp_dpca_wg	PROD_DIR	ASAR DPCA cancellation weights. Product file generated by the RSP. "id" simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	588
"id".ir.rsp_tar_dw	PROD_DIR	ASAR DPCA target deweighting factors. Product file generated by the RSP. "id" simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	768

Table A-4 Product Group Data Files (Page 8 of 8)

File Format	Root Directory	Description	Structure Definition	Max Size (MB)
"id".ir.rsp_repli	PROD_DIR	Weighted frequency-domain pulse compression replicas. Product file generated by the RSP. "id" simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control_block data update intervals	4
"id".dt.detection	PROD_DIR	Detection data. Product file generated by the RSP. "id" simulation run identifier, specified when starting simulation run.	t_csip_header_block t_csip_control block data update intervals	768

Table A-5 Log Group Data Files

File Name	Root Directory	Description	Structure Definition	Max Size (MB)
“id”.lg.gen_err	LOGS_DIR	ASCII error log file for a specified signal generation activity. Created by the Testbed Controller, also includes error log information received from the RSS during signal generation. “id” generation run identifier, specified when starting signal generation.	ASCII error messages	N/A
“id”.lg.sim_err	LOGS_DIR	ASCII error log file for a specified simulation run activity. Created by the Testbed Controller, also includes error log information received from the RSS during signal data playback and from ESP and RSP during signal processing. “id” simulation run identifier, specified when starting simulation run.	ASCII error messages	N/A

A6 Task Implementation

The following sub-sections provide a description of the implementation of each of the Testbed Controller tasks identified in the tasking model diagram of Figure A-1. The task implementation is described in terms of the software packages (CSUs) identified in Section A4.

A6.1 Satellite/Radar Definition Task

Description: The purpose of this task is to support the user selection of satellite, antenna, receiver front-end, antenna beam pointing and transmit waveform characteristics, radar signal simulation data capture and reduction options, as well as simulated radar signal generation options. For more details on these characteristics and options, the reader should refer to the Satellite/Radar HMI table in Appendix A of the Testbed Specification [A-2].

This task is implemented as a stand-alone process which is invoked by the user; each user can have multiple sessions of the same HMI process. The user interface is running in an X Window environment with a hierarchical set of graphical dialog windows (or dialog boxes, the top-level window is referred to as a visualization window, see Appendix G for more details).

These dialog windows are designed using a GUI (Graphical User Interface) builder, UIM/X User Interface Management System (COTS software) with OSF/Motif widget set: push buttons, sliders, option buttons, and text widgets (see A2.4 in Appendix A of the Testbed Preliminary Design Document [R-3]). The UIM/X tool supports the graphical layout design of the dialogue windows and provides capability for the software developer to write/edit and test C code for the creation and display of customized widgets using standard Motif widgets as well as the widget callback functions.

This task calls various initialization routines including routines to display the top-level visualization window of the Satellite/Radar Definition and then enters an infinite loop awaiting X events and never to return with exception that the user selects the "Exit" option to terminate this task. The user can then open a new definition using "New" option or modify an existing definition using the "Open" option.

Each dialog window may contain widgets which support user entry of parameter values or selection of discrete choices or provide a mean to display and activate a lower level dialogue window in the hierarchical window tree structure. All parameter values or choices are initialized by the software to provide default selections for a new definition or when the user changes the data structure of an existing definition through the selection of certain parameter choices such as the Concept parameter. All parameter defaults and valid ranges are stored in a single include file (see crdh_priv.h in Appendix D1.1 and Appendix G for any deviations from HMI table listed in Appendix A of the Testbed Specification [A-2]). Activation of a widget by the user (point and click using a mouse or keyboard numeric data entry) will generate an X event which in turn invokes a callback function.

A callback function may perform range checking on numeric entry (text widget/slider), save the valid data in a temporary widget in-memory data structure and update the dialogue window display with the accepted numeric value. In general, if a numeric entry is out of range, it is set to the nearest range limit. If the numeric entry is non-numeric, the previous numeric value will be retained (default value is always provided). Other callback function may display the new state of the push button or option button or the lower level dialogue window.

Only one dialogue window will be active at a time. In general, a lower level dialogue window is deactivated, removed from the display and control returned to an immediate higher level window through the selection of an "OK" or "Cancel" button widget in the lower level window. If the "OK" button is selected, the callback function will save all the parameter values and choices into an in-memory data structure for the current dialogue window before deactivating it. At the end of a Satellite/Radar definition session, the in-memory data structure is saved into a Satellite/Radar definition file in XDR format through a "Save" or "Save-As" option in the top-level visualization window.

Antenna and receiver front-end characteristics are a reusable subset of the Satellite/Radar HMI parameters. These characteristics are saved during one Satellite/Radar definition session and are reusable at a later time during another session.

The transmit waveform characteristics are defined in terms of waveform parameters which vary with time. A Contiguous Time Interval (CTI) of the transmit waveform is defined by subdividing the time interval into a hierarchy of subintervals (CTI into dwell intervals, dwell interval into NCI intervals, and NCI interval into CPIs) over which certain waveform parameters remain constant. The top-level visualization window supports the random access and parameter entries for these subintervals. It also supports editing functions such as "Create", "Delete", "Copy" function for individual subintervals; for example: create a dwell interval, delete a NCI interval etc.

The antenna beam pointing geometry is changeable from dwell interval to dwell interval and different pointing geometry definition is supported for each dwell.

Structural parameter dependency logic (certain parameter structure is dependent on the choice made for other paramter(s), see Appendix A of the Testbed Specification [A-2]) is implemented to interact with the user through ghosting of certain widgets within an active dialogue window.

Most of the parameter value dependent ranges (also refer to as smart ranges, parameter range which is a function of other parameter value(s) or choice(s), see Appendix A of the Testbed Specification [A-2]) are implemented as fixed ranges independent other parameters. The fixed ranges are chosen to be broad enough to accomodate the narrower parameter value dependent ranges. Validation of parameter value against smart ranges such as the antenna beam pointing geometry whose range is dependent on transmit frequency and certain antenna characteristics, is performed upon user activation. Smart range validation is performed on the entire Satellite/Radar Definition upon user selection of the "Validate" button widget which invokes the services of the CVAL CSU. The CVAL CSU computes the smart ranges by operating on the in-memory version of the Satellite/Radar Definition, checks the Satellite/Radar parameters against these smart ranges and generates a Validation Message Log (see Appendix D1.17 for details). A smart range is provided for the Site parameter automatically, see Appendix G for details.

Most of the parameter value dependent defaults (or smart defaults) provided in the Satellite and Radar HMI table in Appendix A of the Testbed Specification [A-2] are implemented as fixed defaults independent of other parameters with the exception of those which are depending only on the Concept parameter, see Appendix G for details.

When the Satellite/Radar Definition is saved (using "Save" or "Save As" option), CVAL is invoked and the transmit waveform duration (in seconds) is computed and saved with the Satellite/Radar Definition.

For more information on task implementation, the reader should refer to Appendix D1.1, the crdh_main.c file.

Structure: This task is implemented in the package CRDH which has a main program and several source files (.i files) one for each dialogue window, which is generated using the UIM/X tool. This task uses COTS software (X libraries: Xt, Xm, Xlib; Ux) for Motif widget management and window control/display. In addition, it utilizes service functions of the following packages:

CERR -	to format and log error message
CRAD -	to input/output Satellite/Radar Definition file and Antenna Model Definition file
CVAL -	to validate Satellite/Radar Definition
CUIS -	to display commonly used dialogue windows, i.e., confirmation, warning, information, and file selection dialogue windows and to perform common HMI related services

For more information on task structure, the reader should refer to Figure A-3, structure chart of package CRDH.

Interfaces: There are three external interfaces to this task:

The Dialogues interface consists of dialogue windows is described in Appendix G.

The Antenna Model Definition and the Satellite/Radar Definition interface data structures are described in Appendix D1.8.

The Validation Message Log interface data structures are described in Appendix D1.17.

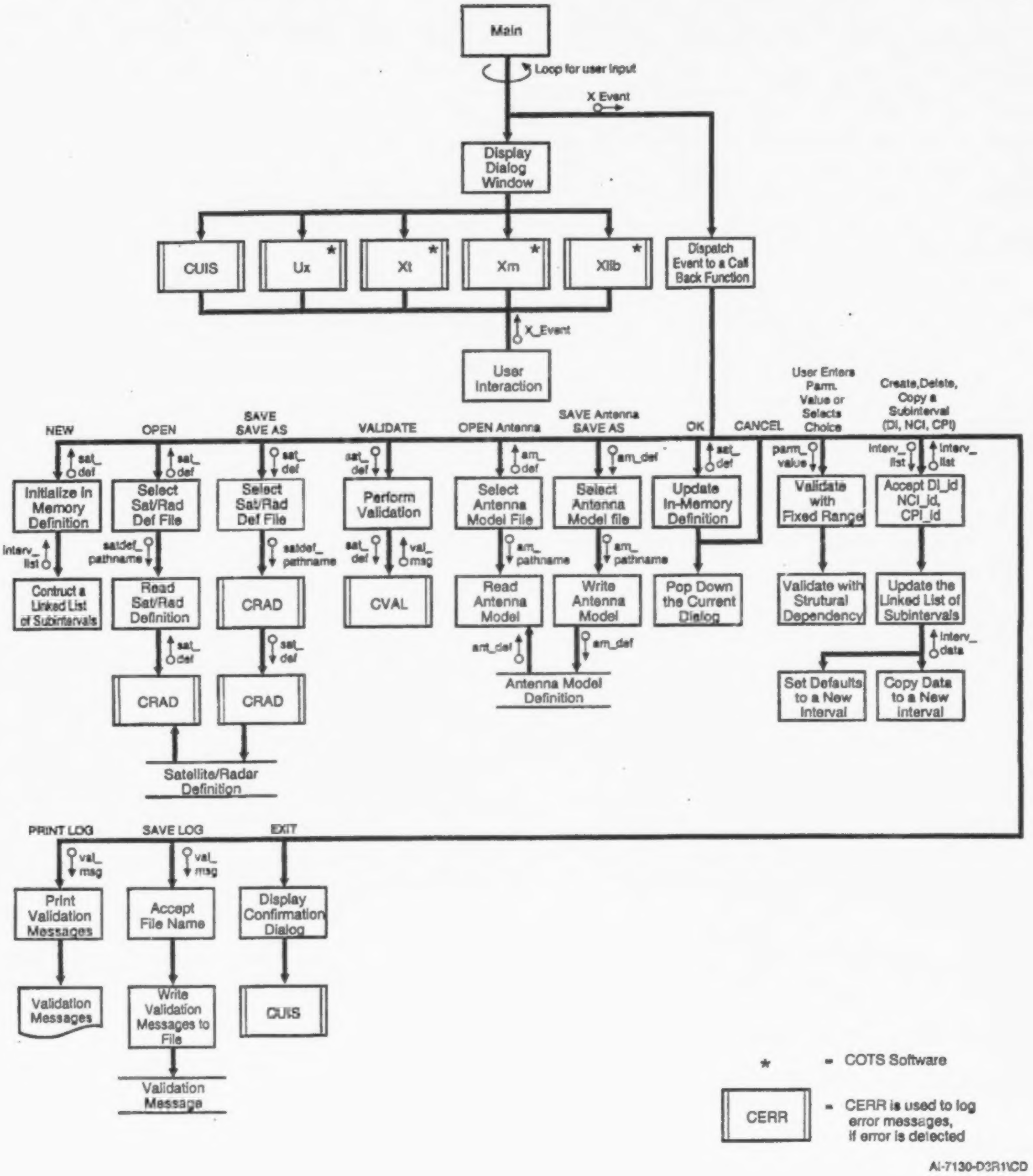

Main
Loop for user input
X Event
Display Dialog Window
CUIS
Ux *
Xt *
Xm *
Xlib *
Dispatch Event to a Call Back Function
X_Event
User Interaction
NEW
OPEN
SAVE
SAVE AS
VALIDATE
OPEN Antenna
SAVE Antenna
SAVE AS
OK
CANCEL
User Enters Parm. Value or Selects Choice
Create,Delete, Copy a Subinterval (DI, NCI, CPI)
sat_def
am_def
parm_value
interv_list
Initialize In Memory Definition
Select Sat/Rad Def File
Select Sat/Rad Def File
Perform Validation
Select Antenna Model File
Select Antenna Model file
Update In-Memory Definition
Validate with Fixed Range
Accept DI_id NCI_id, CPI_id
interv_list
satdef_pathname
val_msg
am_pathname
Contruct a Linked List of Subintervals
Read Sat/Rad Definition
CRAD
CVAL
Read Antenna Model
Write Antenna Model
Pop Down the Current Dialog
Validate with Strutural Dependency
Update the Linked List of Subintervals
ant_def
am_def
interv_data
CRAD
CRAD
Antenna Model Definition
Set Defaults to a New Interval
Copy Data to a New interval
Satellite/Radar Definition
PRINT LOG
SAVE LOG
EXIT
val_msg
Print Validation Messages
Accept File Name
Display Confirmation Dialog
Validation Messages
Write Validation Messages to File
CUIS
Validation Message
* - COTS Software
CERR - CERR is used to log error messages, if error is detected
AI-7130-D3R1\CD

FIGURE A-3 CDRH PACKAGE STRUCTURE CHART

A6.2 Environment Definition Task

A6.2.1 Introduction

The Environment Definition HMI implemented in the CENH CSU provides a user friendly graphical interface for entering and editing jammers, clutter, precipitation cells, etc. The data can be accessed through several levels of popup screens and menus as well as through an FOV (Field of View) display. The FOV display provides a 2 dimensional azimuthal equidistant projection of the earth's surface, initially centred at the sub-satellite point. A zoom feature is available for a more detailed view of a particular area of interest. The currently selected object is highlighted on the FOV display and on the selection list and the associated parameter entry screen is also displayed. A field entry screen will be available for specifying dwell/cpi number for prediction the future position of selected objects. The predicted position information will be displayed in a scrolled window for each object and the predicted beam position will be displayed on the FOV display.

The FOV interactive display is being implemented using PEX (PHIGS Extensions to X) which provides hierarchical display list oriented system with the capability of easily transforming between world and display co-ordinates. The environment definition parameters are being stored in XDR format which provides a platform independent way of handling binary data files.

A6.2.2 CSC/CSU Logic Flow Diagrams

Figure A-4 depicts a high level description of the control flow for the environmental HMI, lower level details were left out of this diagram for the sake of clarity. Figure A-5 gives a more detailed description of the discrete clutter editing interface which is representative of the rest of the interface.

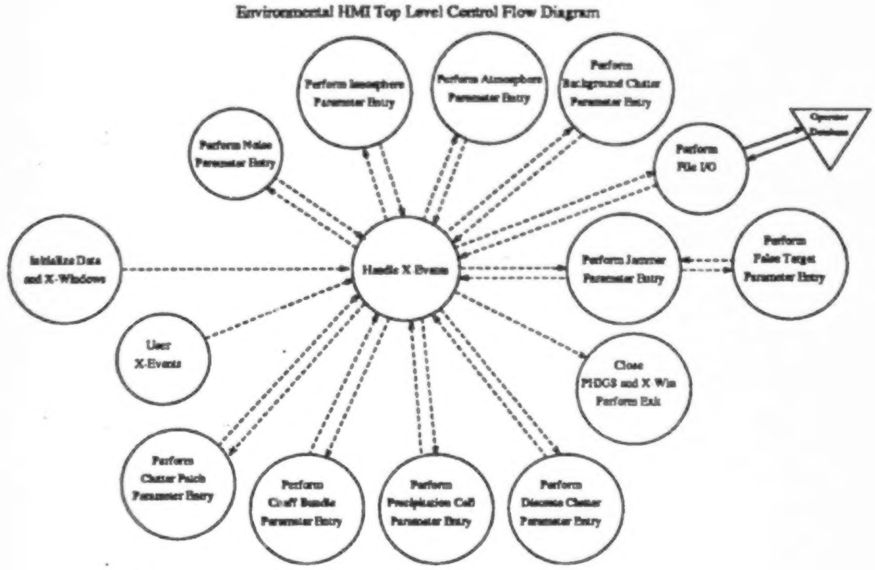

Environmental HMI Top Level Control Flow Diagram
Perform Ionosphere Parameter Entry
Perform Atmosphere Parameter Entry
Perform Background Clutter Parameter Entry
Perform Noise Parameter Entry
Perform File I/O
Operator Database
Initialize Data and X-Windows
Handle X Events
Perform Jammer Parameter Entry
Perform False Target Parameter Entry
User X-Events
Close PHIGS and X Win Perform Exit
Perform Clutter Patch Parameter Entry
Perform Chaff Bundle Parameter Entry
Perform Precipitation Cell Parameter Entry
Perform Discrete Clutter Parameter Entry

FIGURE A-4 ENVIRONMENT DEFINITION HMI TOP LEVEL CONTROL FLOW

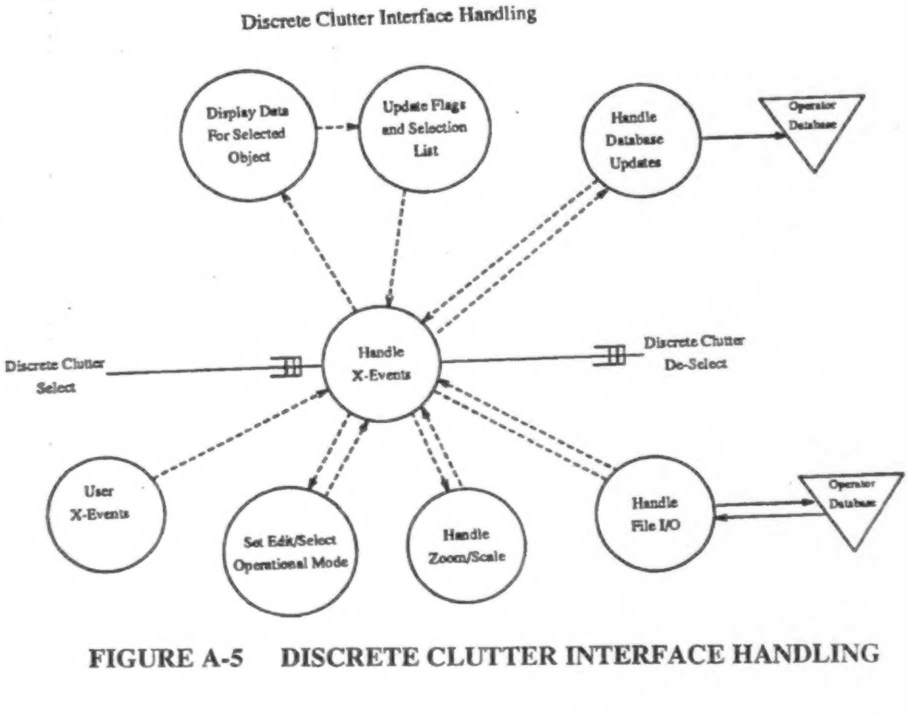

Discrete Clutter Interface Handling
Display Data For Selected Object
Update Flags and Selection List
Handle Database Updates
Operator Database
Discrete Clutter Select
Handle X-Events
Discrete Clutter De-Select
User X-Events
Set Edit/Select Operational Mode
Handle Zoom/Scale
Handle File I/O
Operator Database

FIGURE A-5 DISCRETE CLUTTER INTERFACE HANDLING

A6.2.3 Module Design

A6.2.3.1 Design Logic

The design logic for each module shown in the CENH CSU Module Invocation Tree is contained in Appendix D1.2 together with applicable design reports produced by CDADL.

Module Invocation Hierarchy

See Appendix D1.2 for module invocation hierarchy.

A6.2.3.2 Module Test Plan

TABLE A-6 CENH CSU TESTING MATRIX

Module Name	Desk Inspection	Functional Test	Performance Analysis
f_cenh_main	X		
f_cenh_init	X		
f_cenh_close	X		
f_cenh_handle_lmouse	X		
f_cenh_handle_mmouse	X		
f_cenh_handle_rmouse	X		
f_cenh_empty_fov	X		
f_cenh_load	X		
f_cenh_ld_radar	X		
f_cenh_redraw	X		
f_cenh_save	X		
f_cenh_select_list	X		
f_cenh_valid	X		
f_cenh_edit_object	X		

A6.2.3.2.1 CENH CSU Test Plan

Objective

Verify that the user interface behaves correctly for data field entry as well as for interactive FOV data entry.

Case 1: Ensure that field entries outisde of range bounds are rejected.

Case 2: Ensure that the selection of FOV items results in the appropriate data being displayed in the parameter entry screen as well as the object being highlighted on the FOV screen and on the selection list.

Case 3: Ensure that the zoom feature redraws the objects in eir correct locations.

Case 4: Ensure that the time varying calculations are performed correctly for selected objects.

Objective

Verify that output data is correctly written to and read from XDR formatted files.

A6.3 Simulation Run Definition Task

Description: The purpose of this task is to support the user in specifying how the simulated radar signal data generated is to be processed by the DSP in a simulation run, specifically, definition of a simlulation run involves the selection of DSP unit configurations (ESP and RSP), nulling auxiliaries for each main beam/phase centre/blanker, DSP processing parameters (normal equations, pre-doppler blankers, MASR DPCA, digital PMC, pulse doppler, pulse compression, ASAR DPCA, range-doppler blanking, noncoherent integration, detection and range ambiguity resolution), signal data capture and reduction options, normal equations products and DSP products capture options. For more details on these simulation run parameter characteristics and options, the reader should refer to the Simulation Run HMI table in Appendix A of the Testbed Specification [A-2].

Implementation of the Simulation Run Definition task is in the form of a hierarchical set of dialogue windows created using the UIM/X GUI builder, similar to the implementation of the Satellite/Radar Definition task described in Section A6.1. The top-level visualization window supports the random access and parameter entries for the parameter groups. Each parameter group will appear as a low-level dialogue window for data entries (refer to Appendix G for details).

The user can either open an existing definition using "Open" option or create a new definition using "New" option in the top-level visualization window. At the beginning of a Simulation Run Definition session, the user is asked to select a RSS derived processing parameters file, an output file from a previous signal generation run, as it contains RSS derived DSP processing parameters as well as a pathname of a file containing the Satellite/Radar Definition (an unmodifiable version which is permanently associated with the RSS derived processing parameters file) which the Simulation Run Definition has a dependency on (see Inputs column of Simulation Run HMI table in Appendix A of the Testbed Specification [A-2]). This subset of the Satellite/Radar Definition parameters is used for determining ranges, defaults and structures of some of the parameters found in the Simulation Run Definition. These dependencies are referred to as external dependencies to distinquish them from internal dependencies which exist among Simulation Run Definition parameters. For a new definition, the Satellite/Radar Definition is copied from the RSS derived processing parameters file associated file and stored into another file which is permanently associated with the Simulation Run Definition to facilitate consistency checking between existing Simulation Run Definition and a different RSS derived processing parameters file subsequently selected.

An external structural dependency which is not obvious from the Simulation Run HMI table in Appendix A of the Testbed Specification [A-2] is the requirement that the number of various subintervals (dwell interval, NCI interval and CPI) in the Simulation Run Definition must be the same as the number defined in the Satellite/Radar Definition such that different values or choices for certain DSP processing parameters are selected for different subintervals. The interval (Run) or subinterval of changeability for the Simulation Run parameters are provided in Table 3–4, DSP Processing Parameters, DSP/Testbed Interface Detailed Design [R-1].

In general, all external and internal structural parameter dependencies described in the Simulation Run HMI table in Appendix A of the Testbed Specification [A-2] are implemented in the design of this task.

Most external and internal parameter value dependent ranges are implemented as fixed ranges. The fixed ranges are chosen to be broad enough to accomodate the narrower parameter value dependent ranges. Validation of parameter values against smart ranges such as the snapshot smoothing period whose range is dependent on the number of adaptive weights and sampling rate, is performed upon user activation. Parameter values validation against smart ranges is performed on the entire Simulation Run Definition upon user selection of the "Validate" button widget which invokes the services of the CVAL CSU. The CVAL CSU computes all smart ranges using parameter values stored in the Simulation Run Definition in-memory data structure, checks the Simulation Run parameters against these smart ranges and generates a Validation Message Log. Other smart ranges including most of the DSP unit configuration and nulling systems parameters are provided automatically, see Appendix G for details.

Most of the external and internal parameter value dependent defaults provided in the Simulation Run HMI table in Appendix A of the Testbed Specification [A-2] are implemented as fixed defaults with the exception of those which are depending only on the Satellite/Radar HMI Concept parameter, see Appendix G for details.
If the user opens an existing Simulation Run Definition file or when a different RSS derived processing parameters file is selected, a

consistency check between the Simulation Run Definition and the RSS derived processing parameters file is performed automatically using the sevices of the CVAL CSU. The Satellite/Radar parameters assoicated with the RSS derived processing parameters file are read into an in-memory data structure. The CVAL CSU is then invoked to check the Satellite/Radar parameters (Number of DPCA Groups and Number of Phase Centres per DPCA Group) associated with the Simulation Run Definition and those assoicated with the RSS derived processing parameters file, as well as the Simulation Run parameters whose data structures are dependent on the Satellite/Radar parameters against those Satellite/Radar parameters associated with the RSS derived processing parameters file for consistency. If inconsistency exists, the user will be asked to either select a different RSS derived processing parameters file or proceed with a new Simulation Run Definition (in this case, the existing definition is replaced by a new definition containing default parameter values and choices, the associated Satellite/Radar Definition is also replaced).

As indictated in the above descriptions, there are two types of validation checks performed by the CVAL CSU, i.e., range checking and data structure consistency checking. To simplify implementation of the CVAL CSU, both types of validity checks are performed upon invocation.

For more information on task implementation, the reader should refer to Appendix D1.3, the csdh_main.c file. All parameter defaults and valid ranges are stored in a single include file (see csdh_priv.h in Appendix D1.3).

Structure: This task is implemented in the package CSDH which has a main program and serveral source files (.i files) one for each dialogue window, which are generated using the UIM/X tool. This task uses COTS software (X libraries: Xt, Xm, Xlib; Ux) for Motif widget management and window control/display. In addition, it utilizes the following service functions of the following packages:

CERR -	to format and log error message
CSID -	to input/output Simulation Run Definition file
CGEP -	to input RSS derived processing parameters file
CVAL -	to validate Simulation Run Definition as well as its consistency with Satillite/Radar Definition
CUIS -	to display commonly used dialogue windows, i.e., confirmation, warning, information, and file selection dialogue windows and to perform common HMI related services

For more information on task structure, the reader should refer to Figure A-6, structure chart of package CSDH.

Interfaces: There are four external interfaces to this task:
The Dialogues interface consists of dialogue windows is described in Appendix G.

The Simulation Run Definition interface data structures are described in Appendix D1.10.

The RSS derived processing parameters interface data structures are described in the cgep_intf2.x file listed in Appendix D1.12.

The Validation Message Log interface data structures are described in Appendix D1.17.

Monitor/Control Generation Task

Description: The purpose of this task is to support the user control (initiation and termination) of signal generation activity as well as to support the generation progress monitoring function.

This task is implemented as a hierarchy of dialogue windows, see Appendix G for details.

The user chooses a Satellite/Radar Definition and Environment Definition previously defined by the user through a file selection dialogue window. The user must give each generation run an identifier ("GEN-IDENT"). All output files related to this generation activity including Simulated Radar Signal Data (SRSD) file, RSS derived processing parameter file, RSS intermediate product files and error log file will have this identifier as a file prefix (see Tables A-2 to A-5 for more details on the file naming conventions).

The Monitor/Control Generation Task reads the Satellite/Radar Definition and the Environment Definition files and stores the parameters in an in-memory data structure. Validation check is automatically performed on this in-memory data structure using services provided by the CVAL CSU. If any error is detected, the Validation Message Log will be displayed which can then be optionally printed or saved to a file. In this case, the user will have the option to start the signal generation run using the erroneous parameters or abort the run.

This task is implemented as a client to the Generation Server task and thus the Generation Server must be running before any signal generation execution. Multiple sessions of this HMI task can be attached to the Generation Server. This task issues a "Start" or "Abort" request to the Generation Server via RPC call to start a generation or abort the current generation activity. Since signal generation activity requires the dedicated use of the RSS co-processor, only one generation activity can be activated at a time. Thus the run request will be denied by the Generation Server, if a generation activity is currently executing. During a generation run, this task reports execution progress by displaying a percent completion and possible error indication reported by the RSS Generation Manager and by the Generation Server tasks.

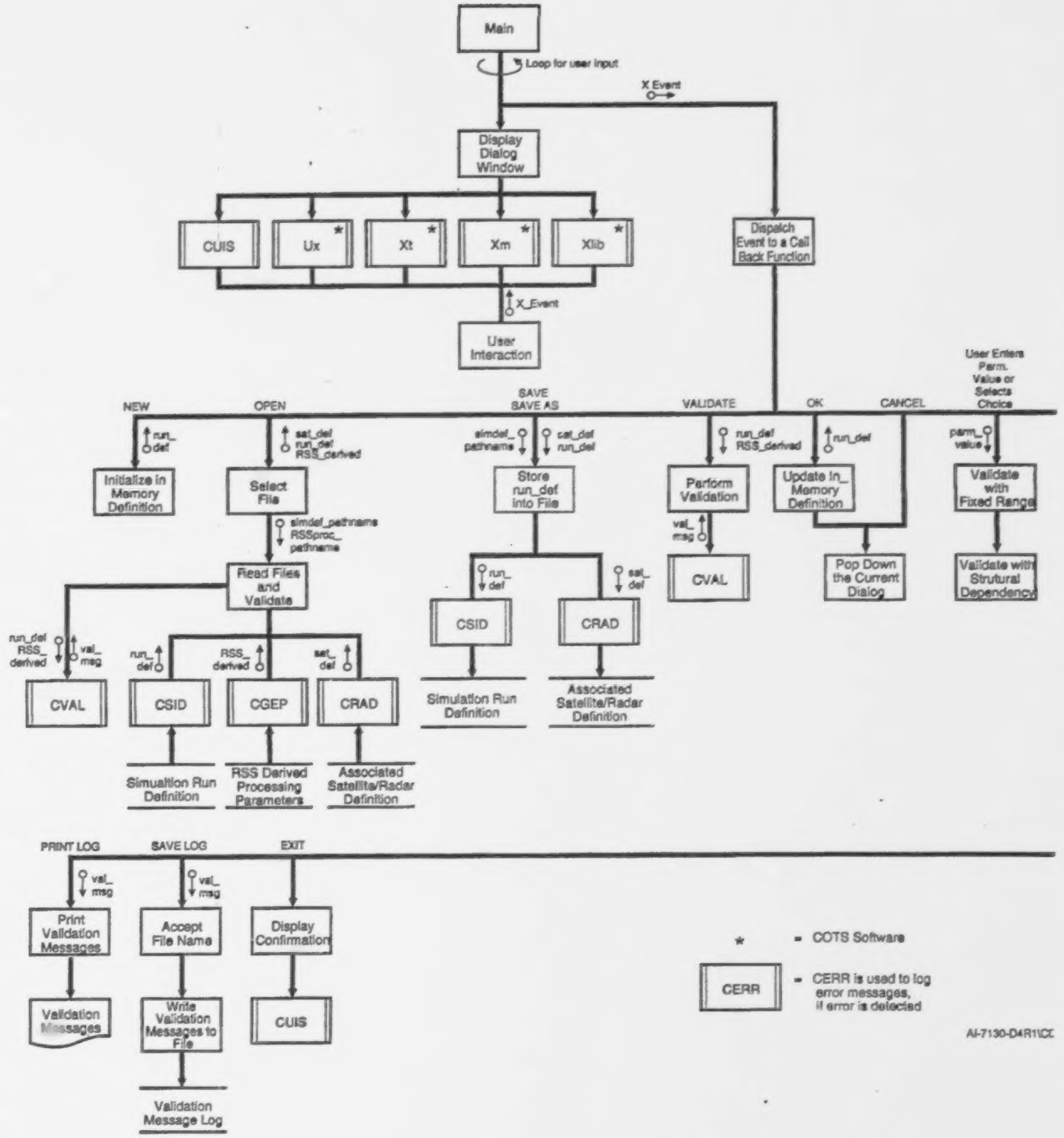

Main
Loop for user input
X Event
Display Dialog Window
CUIS
Ux *
Xt *
Xm *
Xlib *
X_Event
User Interaction
Dispatch Event to a Call Back Function
NEW
OPEN
SAVE
SAVE AS
VALIDATE
OK
CANCEL
User Enters Parm. Value or Selects Choice
run_def
sat_def
run_def
RSS_derived
simdef_pathnams
sat_def
run_def
run_def
RSS_derived
run_def
parm_value
Initialize in Memory Definition
Select File
simdef_pathname
RSSproc_pathname
Store run_def into File
Perform Validation
Update In_Memory Definition
Validate with Fixed Range
val_msg
CVAL
Pop Down the Current Dialog
Validate with Strutural Dependency
Read Files and Validate
run_def
sat_def
CSID
CRAD
run_def
RSS_derived
val_msg
run_def
RSS_derived
sat_def
CVAL
CSID
CGEP
CRAD
Simulation Run Definition
Associated Satellite/Radar Definition
Simualtion Run Definition
RSS Derived Processing Parameters
Associated Satellite/Radar Definition
PRINT LOG
SAVE LOG
EXIT
val_msg
val_msg
Print Validation Messages
Accept File Name
Display Confirmation
Validation Messages
Write Validation Messages to File
CUIS
Validation Message Log
* = COTS Software
CERR = CERR is used to log error messages, if error is detected
AI-7130-D4R1\CC

FIGURE A-6 CSDH PACKAGE STRUCTURE CHART

The generation status is passed to the Monitor/Control Generation task by the Generation Server task using the XChangeProperty function, an X Window System service. The XChangeProperty function is called by the Generation Server task with the generation status passed to it as an argument. The XChangeProperty function generates an X event of type "PropertyNotify" and provides the generation status data structure. The Monitor/Control Generation task is waiting for the occurrence of an X event which triggers the execution of a callback function. In this case, it is a callback function which displays the generation status information.

For more information on task implementation, the reader should refer to Appendix D1.13, the cgeh_main.c file.

Structure: This task is implemented in the package CGEH which has a main program and source files for dialogue windows. This task uses COTS software (X libraries: Xt, Xm, Xlib; Ux) for Motif widget management and window control/display and RPC package for inter-task communications. In addition, it utilizes service functions of the following packages:

CERR -	to format and log error message
CRAD -	to input Satellite/Radar Definition file
CEND -	to input Environment Definition file
CVAL -	to validate Satellite/Rader Definition and Environment Definition parameters
CUIS -	to display commonly used dialogue windows, i.e., confirmation, warning, information, status and file selection dialogue windows and to perform common HMI related services
CGET -	to send RPC message to the Generation Server task

For more information on task structure, the reader should refer to Figure A-7, structure chart of package CGEH.

Interfaces: There are five external interfaces to this task:

The Dialogues interface consists of dialogue windows is described in Appendix G.

The Satellite/Radar Definition interface data structures are described in Appendix D1.8.

The Environment Definition interface data structures are described in Appendix D1.9.

The RPC message interface data structures for messages sent to the Generation Server are described in Appendix D1.14.

The Validation Message Log interface data structures are described in Appendix D1.17.

A6.4 Generation Server Task

Description: The purpose of this task is to support the communications between the Controller and the RSS. This task communicates with the RSS Generation Manager and Monitor/Control Generation tasks. It receives "Start" or "Abort" requests from the Monitor/Control Generation task and then passes them to the RSS Generation Manager. On the reverse direction, it receives responses and generation status (percent completion and error condition) from the RSS Generation Manager. It then passes the generation status including any error conditions detected by itself to the Monitor/Control Generation task. The RPC package (COTS software) is utilized for most inter-task communications except for the passing of the generation status as described in A6.4.

This task acts as a server to the Monitor/Control Generation task and is implemented as a daemon (a continuous background process). This task also acts as a guard, which either grants or denies a generation request sent from one or more instances of the Monitor/Control Genration task (currently only one generation process is allowed in the RSS). When communicating with the RSS Generation Manager task, this task acts as a client to the RSS Generation Manager.

Due to the client/server architecture, this task must be launched before the user can activate the Monitor/Control Generation task. Normally, the SPD system manager (a special user) is in charge of startup and shutdown of the daemons. This task will be launched or terminated by a SPD startup or shutdown script, which invokes a start or shutdown program provided by the package CSYS (see Appendix D1.19, csys_startgen.c and csys_shutdwngen.c for details).

This task is also responsible for logging of its own software errors into an error log file and will merge this error log file with the error log file generated by the RSS Manager for each generation run, see Appendix A7 for details on error handling.

For more information on task implementation, the reader should refer to Appendix D1.14, the cget_main.c file.

Structure: This task is implemented in the package CGET which has a main program. This task uses COTS software (RPC package) for inter-task communications. In addition, it utilizes service functions of the following packages:

CERR -	to log error message and to merge error log files
CGEH -	to send generation status to the Monitor/Control Generation task.
CRGI -	to send and receive RPC messages to and from the RSS Generation Manager task.

For more information on task structure, the reader should refer to Figure A-8, structure chart of package CGET.

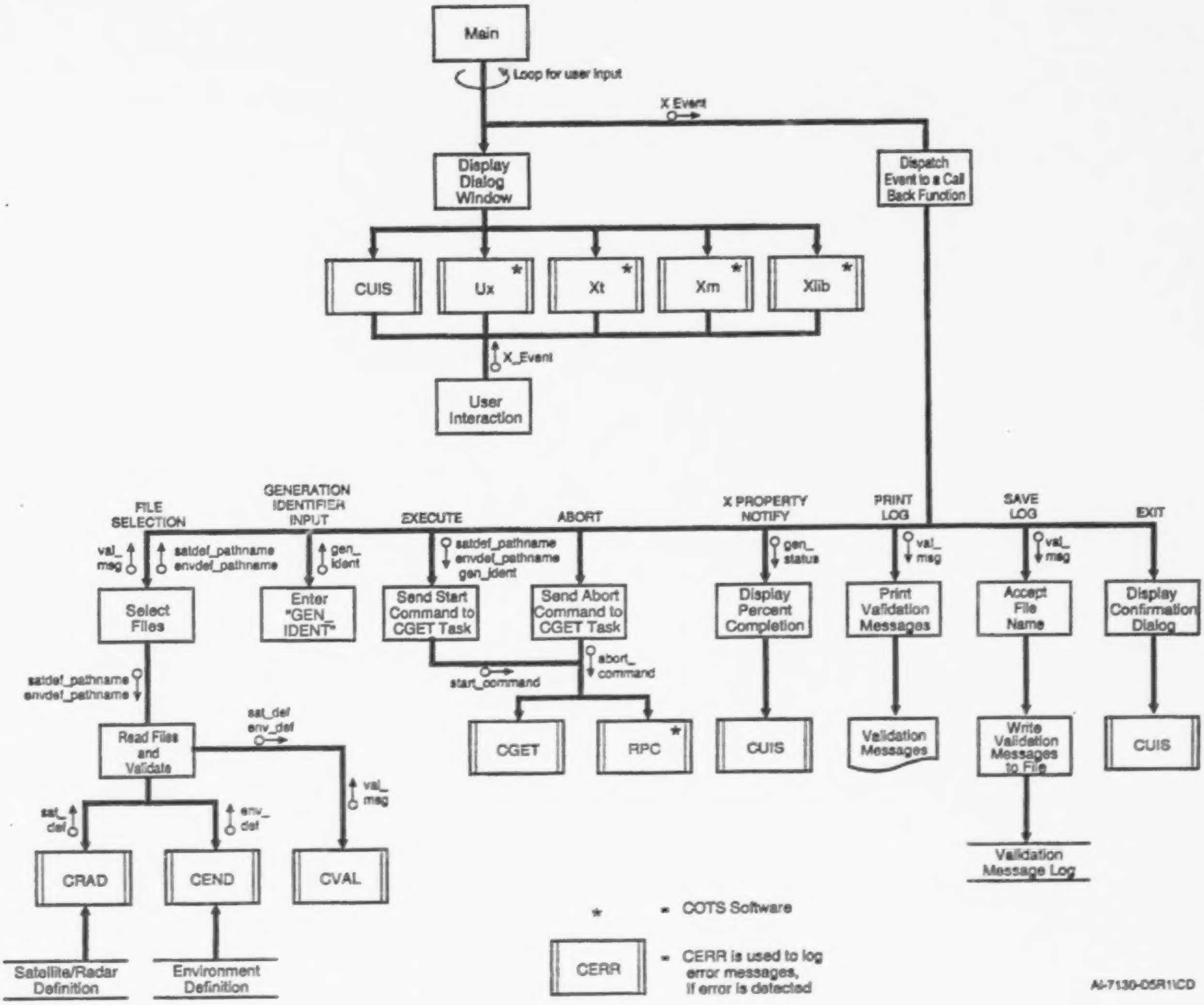

Main
Loop for user input
X Event
Display Dialog Window
Dispatch Event to a Call Back Function
CUIS
Ux *
Xt *
Xm *
Xlib *
X_Event
User Interaction
FILE SELECTION
GENERATION IDENTIFIER INPUT
EXECUTE
ABORT
X PROPERTY NOTIFY
PRINT LOG
SAVE LOG
EXIT
val_ msg
satdef_pathname envdef_pathname
gen_ ident
satdef_pathname envdef_pathname gen_ident
gen_ status
val_ msg
val_ msg
Select Files
Enter "GEN_ IDENT"
Send Start Command to CGET Task
Send Abort Command to CGET Task
Display Percent Completion
Print Validation Messages
Accept File Name
Display Confirmation Dialog
satdef_pathname envdef_pathname
start_command
abort_ command
Read Files and Validate
sat_def env_def
val_ msg
CGET
RPC *
CUIS
Validation Messages
Write Validation Messages to File
CUIS
sat_ def
env_ def
CRAD
CEND
CVAL
Validation Message Log
Satellite/Radar Definition
Environment Definition
* = COTS Software
CERR
= CERR is used to log error messages, if error is detected
AI-7130-05R1\CD

FIGURE A-7 CGEH PACKAGE STRUCTURE CHART

Interfaces: There are three external interfaces to this task:

The RPC message interface data structures between this task and the RSS Generation Manager is described in Appendix D1.4.

The generation status data structures sent to the Monitor/Control Generation task are described in Appendix D1.13

The Error log interface data structures are described in Appendix D1.18.

A6.5 Monitor/Control Simulation Task

Description: The purpose of this task is to support the user control (initiation and termination) of simulation run activity as well as to support the simulation progress monitoring function.

This task is implemented as a hierarchy of dialogue windows, see Appendix G for details.

The user chooses the RSS derived processing parameter file and Simulation Run Definition through a file selection dialogue (note that both the Simulated Radar Signal Data (SRSD) file and the derived processing parameter file are output in pair from a signal generation activity, i.e., both have the same generation identifier "GEN-IDENT" and hence, selecting one automatically selects the other). In addition,m the user must provide each run with an identifier ("RUN-IDENT"). All output files related to this simulation run activity including DSP parameter file, RSP detection file, ESP and RSP intermediate product file2 and error log file will have this identifier as a file prefix (see Tables A-1 to A-5 for details on file naming conventions).

The Monitor/Control Simulation task reads the Simulation Run Definition and the RSS derived processing parameters files and stores the parameters in an in-memory data structure. Validation check is automatically performed on this in-memory data structure using services provided by the CVAL CSU. If any error is detected, the Validation Message Log will be displayed which can then be optionally printed or saved to a file. The user will then be presented the option of starting the simulation run using the erroneous parameters or to abort the simulation run.

This task is implemented as a client to the Simulation Server task and thus the Simulation Server must be running before any simulation run execution. This task issues a "Start" or "Abort" request to the Simulation Server via RPC call to start a simulation or abort the current simulation activity. The run request will be denied by the Simulation Server, if there is a simulation activity already executing, since only one simulation run can take place at a time (requires dedicated use of the RTSG and DSP).

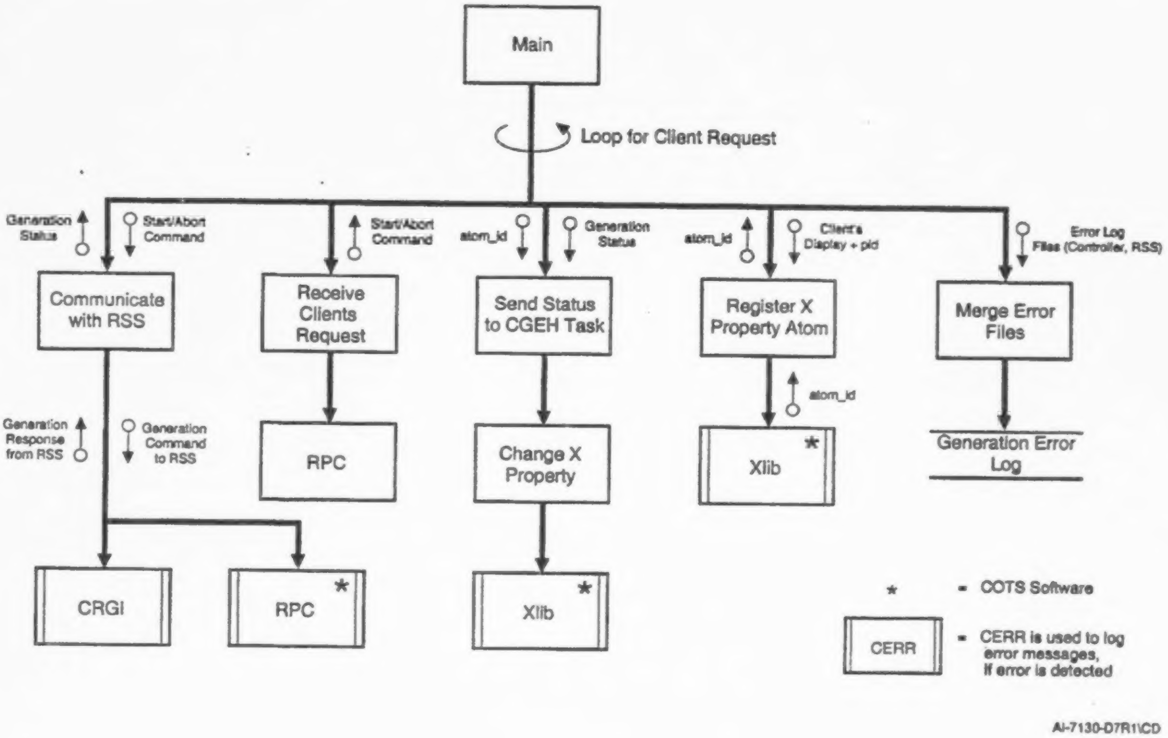

Main
Loop for Client Request
Generation Status
Start/Abort Command
Start/Abort Command
atom_id
Generation Status
atom_id
Client's Display + pid
Error Log Files (Controller, RSS)
Communicate with RSS
Receive Clients Request
Send Status to CGEH Task
Register X Property Atom
Merge Error Files
Generation Response from RSS
Generation Command to RSS
RPC
Change X Property
atom_id
Xlib *
Generation Error Log
CRGI
RPC *
Xlib *
* - COTS Software
CERR - CERR is used to log error messages, if error is detected
AI-7130-D7R1\CD

FIGURE A-8 CGET PACKAGE STRUCTURE CHART

During a simulation run, the following simulation status information is received by the Monitor/Control Simulation task from the Simulation Server task:

- real-time rate factor reported on a per simulated radar signal data segment iteration basis;
- percent completion reported on a per detection message reception basis;
- any error conditions reported by the RSS, ESP and RSP software as well as by the Simulation Server task.

The simulation status information is sent to the Monitor/Control Simulation task by the Simulation Server task using the XChangeProperty function, an X Window System service. The XChangeProperty function is called by the Simulation Server task with the simulation status passed to it as an argument. The XChangeProperty function generates an X event of type "PropertyNotify" and provides the simulation status data structure . The Monitor/Control Simulation task is waiting for the occurrence of an X event which triggers the execution of a callback function. In this case, it is a callback function which displays the simulation status information.

For more information on task implementation, the reader should refer to Appendix D1.15, the csih_main.c file.

Structure: This task is implemented in the package CSIH which has a main program and source files for dialogue windows. This task uses COTS software (X libraries: Xt, Xm, Xlib; Ux) for Motif widget management and window control/display and RPC package for inter-task communications. In addition, it utilizes service functions of the following packages:

CERR -	to format and log error message;
CSID -	to input Simulation Run Definition file;
CGEP -	to input the RSS derived processing parameters file;
CVAL -	to validate the Simulation Run Definition;
CUIS -	to display commonly used dialogue windows, i.e., confirmation, warning, information, status and file selection dialogue windows and to perform common HMI related services;
CSIT -	to send RPC message to the Simulation Server task.

For more information of task structure, the reader should refer to Figure A-9, structure chart of package CSIH.

Interfaces: There are five external interfaces to this task:

The Dialogues interface consists of dialogue windows is described in Appendix G.

The Simulation Run Definition interface data structures are described in Appendix D1.10.

The RSS derived processing parameter interface data structures are described in the cgep_intf2.x file listed in Appendix D1.12.

The RPC message interface data structures sent to the Simulation Server are described in Appendix D1.16.

The Validation Log interface data structures are described in Appendix D1.17.

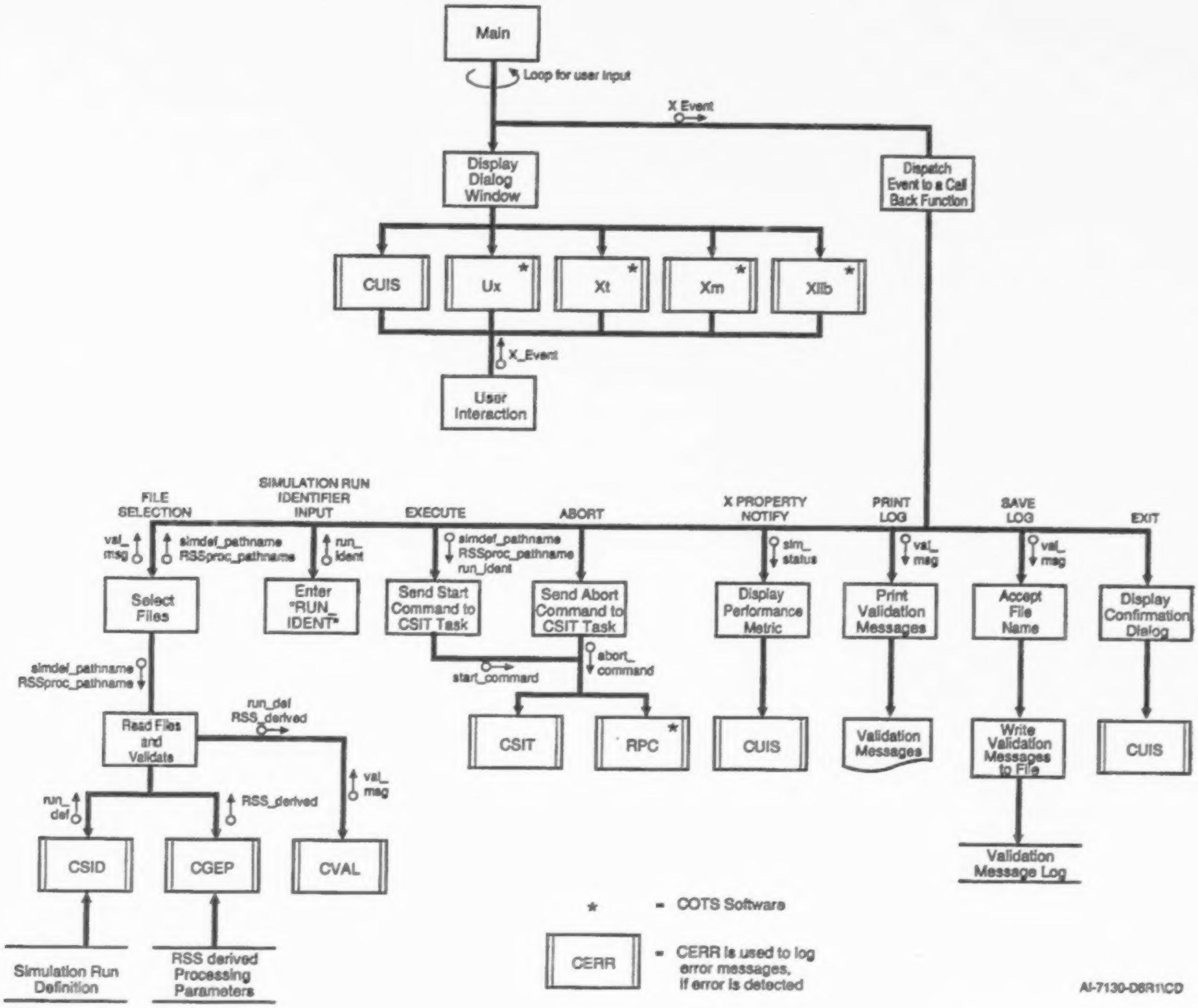

Main
Loop for user input
X Event
Display Dialog Window
Dispatch Event to a Call Back Function
CUIS
Ux *
Xt *
Xm *
Xlib *
X_Event
User Interaction
FILE SELECTION
SIMULATION RUN IDENTIFIER INPUT
EXECUTE
ABORT
X PROPERTY NOTIFY
PRINT LOG
SAVE LOG
EXIT
val_ msg
simdef_pathname RSSproc_pathname
run_ ident
simdef_pathname RSSproc_pathname run_ident
sim_ status
val_ msg
val_ msg
Select Files
Enter "RUN_ IDENT"
Send Start Command to CSIT Task
Send Abort Command to CSIT Task
Display Performance Metric
Print Validation Messages
Accept File Name
Display Confirmation Dialog
simdef_pathname RSSproc_pathname
start_command
abort_ command
run_def RSS_derived
Read Files and Validate
CSIT
RPC *
CUIS
Validation Messages
Write Validation Messages to File
CUIS
run_ def
RSS_derived
val_ msg
CSID
CGEP
CVAL
Validation Message Log
Simulation Run Definition
RSS derived Processing Parameters
* = COTS Software
CERR = CERR is used to log error messages, if error is detected
AI-7130-D6R1CD

FIGURE A-9 CSIH PACKAGE STRUCTURE CHART

A6.6 Simulation Server Task

Description: The purpose of this task is to support the communications between the Controller and the RSS as well as between the Testbed and the DSP units (ESP and RSP).

This task acts as the server to the Monitor/Control Simulation task and is implemented as a daemon (a continuous background process). This task also acts as a guard, which either grants or denies a simulation request sent from one or more instances of the Monitor/Control Simulation task (only one simulation run is allowed at a given time).

Upon receiving a "Start" or "Abort" request from the Monitor/Control Simulation task via RPC call, this task communicates with the RSS Playback Manager, ESP Manager, and RSP Manager tasks by sending the "Run" ("Start Playback" for the RSS) or "Stop" ("Abort Playback" for the RSS) commands and receiving responses from them. For a "Start" request, the control sequence involves issuing the "Configure" (or "Update Parameters" followed by "Run") command to the RSP and the ESP and wait for command completion acknowledge messages received from both the RSP and ESP Manager tasks before sending the "Start Playback" command to the RSS Playback Manager task. For an "Abort" request, the order of the control sequence is reversed.

Both the "Update Parameters" and the "Configure" commands sent to the DSP units will contain the DSP Parameters data structure. The Simulation Server task reads the Simulation Run Definition, the RSS derived processing parameters file and its associated Satellite/Radar Definition into an in-memory data structure. The DSP Parameters are generated from a subset of this in-memory data structure. A complete set of DSP Parameters are also written out as a DSP parameter file (file name contains the "RUN-IDENT" prefix), which will be used subsequently by the Data Analysis task in performing the parameter estimation function.

The Simulation Server task receives the detection messages containing detection data from the RSP and writes them into a file (file name contains the "RUN-INDENT" prefix). In addition, it generates the simulation status information from the detection messages as well as from unsolicited status messages received from the RSS Playback Manager, ESP Manager, and RSP Manager tasks. The simulation status information includes the following:

- real-time rate factor reported on a per simulated radar signal data segment iteration basis;
- percent completion reported on a per detection message reception basis;
- any error conditions reported by the RSS, ESP and RSP software as well as by the Simulation Server task itself.

During a simulation run, a simulation progress status RPC message (contains possible error condition and elapsed time in seconds measured from the start of a simulation run to the end of the current simulated radar signal data iteration) is sent by the RSS Playback Manager task (sent at the end of each iteration for elapsed time and at any time for error condition). A real-time rate factor is computed from the ratio of the transmit waveform duration (extracted from the RSS derived processing parameter file) to the average iteration elapsed time.

A percent completion status is computed by the Simulation Server task and sent to the Monitor/Control Simulation task every time a detection message (RPC) is received by the Simulation Server task. The number of NCI intervals in a simulation run is computed at the beginning of processing a "Run" request during the generation of the DSP Parameters. It is the product of the number of NCI intervals per data iteration (computed from the in-memory Simulation Run Definition) and the number of data iterations in the simulation run (a part of the "Start" request message). A current NCI interval count is also maintained and updated every time a detection message (contains Dwell and NCI identifiers) is received. The percent completion is computed from the ratio of the current NCI interval count to the total number of NCI intervals.

Other error conditions are reported by the ESP and RSP Manager tasks via the "Unit Error" message as well as by the Simulation Server task itself.

Most inter-task communications are performed via RPC calls with the exception of passing of the simulation status to the Monitor/Control Simulation task as described in A6.6.

Due to the client/server architecture, this task must be launched before the user can activate the Monitor/Control Simulation task. Normally, the SPD system manager (a special user) is in charge of startup and shutdown the daemons. This task will be launched or terminated by a SPD startup or shutdown script, which invokes a startup or shutdown program provided by the package CSYS (see csys_startsim.c and csys_shutdwnsim.c for details).

This task is also responsible for logging of its own software errors into an error log file and will merge this error log file with the error log files generated by the RSS Playback Manager, the ESP Manager and the RSP Manager tasks for each simulation run, see Appendix A7 for details on error handling.

For more information on task implementation, the reader should refer to Appendix D1.16, the csit_main.c file.

Structure: This task is implemented in the package CSIT which has a main program. This task uses COTS software (X libraries) for sending progress to the Monitor/Control Generation task as an X atom and RPC package for inter-task communications between this task and the RSS Generation Manager. In addition, it utilizes service functions of the following packages:

CERR - to log error message and to merge error log files;

CSID -	to input Simulation Run Definition file;
CGEP -	to input the RSS derived processing parameters file;
CSIH -	to send simulation status to the Monitor/Control Simulation task;
CDSI -	to generate DSP Parameters and to output them to a file, to communicate with the ESP and RSP Manager tasks, and to receive Detection messages from the RSP Manager task;
CRPI -	to communicate with the RSS Playback Manager task;
CSIP -	to write the Detection messages received from the RSP Manager task into a file.

For more information of task structure, the reader should refer to Figure A-10, structure chart of package CSIT.

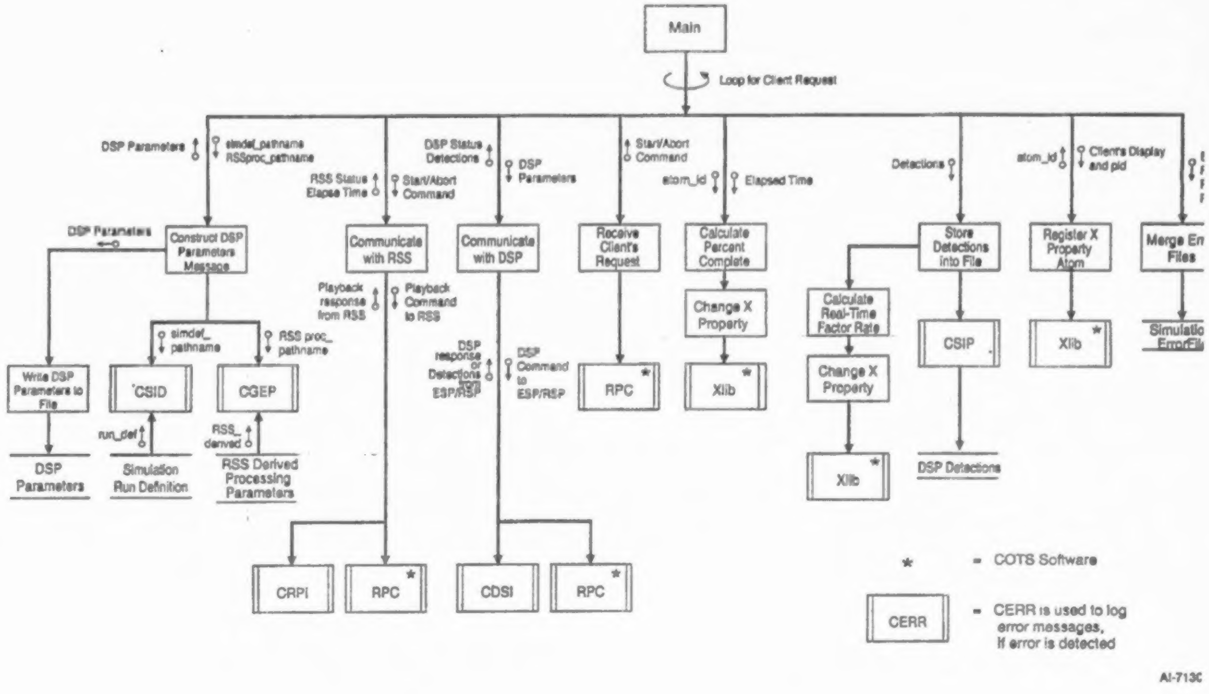

Main
Loop for Client Request
DSP Parameters
simdef_pathname
RSSproc_pathname
RSS Status
Elapse Time
Start/Abort
Command
DSP Status
Detections
DSP
Parameters
Start/Abort
Command
atom_id
Elapsed Time
Detections
atom_id
Client's Display
and pid
DSP Parameters
Construct DSP
Parameters
Message
Communicate
with RSS
Communicate
with DSP
Receive
Client's
Request
Calculate
Percent
Complete
Store
Detections
into File
Register X
Property
Atom
Merge Er
Files
Playback
response
from RSS
Playback
Command
to RSS
simdef_
pathname
RSS proc_
pathname
Change X
Property
Calculate
Real-Time
Factor Rate
CSIP
Xlib *
Simulatio
ErrorFil
Write DSP
Parameters to
File
CSID
CGEP
DSP
response
or
Detections
from
ESP/RSP
DSP
Command
to
ESP/RSP
RPC *
Xlib *
Change X
Property
run_def
RSS_
derived
DSP
Parameters
Simulation
Run Definition
RSS Derived
Processing
Parameters
Xlib *
DSP Detections
CRPI
RPC *
CDSI
RPC *
* = COTS Software
CERR = CERR is used to log
error messages,
if error is detected
AI-713C

FIGURE A-10 CSIT PACKAGE STRUCTURE CHART

Interfaces: There are eight external interfaces to this task:

The Simulation Run Definition interface data structures are described in Appendix D1.10.

The RSS derived processing parameter interface data structures are described in the in the cgep_intf2.x file listed in Appendix D1.12.

The simulation status data structures sent to the Monitor/Control Simulation task are described in Appendix D1.15

The RPC message interface data structures between this task and the RSS Playback Manager is described in the package CRPI in Appendix D1.5.

The RPC message interface data structures between this task and the ESP or RSP Manager is described in Appendix D1.6.

The DSP Parameters file data structure is described in Appendix D1.6

The Detection file data structure is described in Appendix D1.11

The Error log interface data structures are described in Appendix D1.18.

A6.7 Data Analysis Task

Description: The purpose of this task is to support the signal data analysis operations on the RSS/ESP/RSP intermediate products or DSP detection data.

This task is implemented using the PV-Wave visual data analysis software package supplied by Precision Visuals Inc. (the company has recently been changed to Visual Numerics Inc.).

Macros written in PV-Wave command language are to be provided to perform the data ingest function for intermediate products, detection data and the DSP processing parameters as well as the parameter estimation function for the detection data.

The parameter estimation function uses the ambiguous range detection data and the DSP processing parameters as inputs and generates unambiguous range detection data (estimates of unambiguous target range, target velocity, target look angles and RCS).

For more details on the implementation of the data ingest and parameter estimation functions, the reader should refer to Appendix D1.7.

Structure: Not provided by Commercial Off The Shelf (COTS) software vendor.

Interfaces: There are three interfaces to this package:

The RSS intermediate products data structures are described in the cgep_intf1.x file of package CGEP listed in Appendix D1.12.

The detection data and DSP intermediate products data structures are described in the package CSIP in Appendix D1.11.

The DSP processing parameters data structures are described in the CDSI package listed in Appendix D1.6.

A7 Error Handling

This section defines the guidelines to be followed in designing and implementing the software error handling within the SPD software. These guidelines apply to all developed software, including software for the Sun SPARCStation, DSP executive controller and VASP processing elements.

A7.1 Objectives

Given that the SPD is a laboratory-based system for the demonstration and validation of radar signal processing algorithms, there is no need for implementation of an error handling scheme to provide for fault-tolerant operation. As such, the objectives for an error handling scheme for the SPD will be as follows:

a) detect serious error conditions and prevent system from continuing thus generating unpredictable results;

b) identify where problem was detected and condition that generated the error;

c) capture additional diagnostic information to aid the user in subsequent analysis;

d) format the error diagnostic information in a consistent manner for ease of analysis;

e) output the error diagnostic information to a known errorlog and provide access for the user to display and analyze the diagnostic information.

A7.2 Definitions

For the SPD developed software, two different categories of errors are considered:

a) status errors
b) run-time errors

Status errors are not true errors but are errors which occur in the normal course of operation (i.e., end of file reached in a file read operation, validation routine finds an invalid value entered by operator, etc.). These errors are not fatal and are also not logged to any error log file.

Run-time errors are true error conditions and are used to indicate software failure where the requested operation cannot be completed due to an erroneous event. Examples of run-time errors are arithmetic errors, I/O errors, etc. Although these errors are not expected to occur in the normal course of operation, an error handling scheme must be implemented to detect and handle these error conditions. These errors are usually fatal and are logged to an error log file.

A7.3 Propagation

A lower level function detecting an error condition may want to return a specific error condition to the calling function plus optionally log the error to the error log. In the case where the low level function is a math function, or data base access function, or other common or utility function, it is sometimes not desirable to log the error condition to the error log but only return the error condition to the calling function. This error propagation and optional logging would continue until the level is reached where the final decision is made on how to handle the error condition. On propagating errors to a higher level function, the software may want to change the error condition to be more meaningful or more specific to the current function.

Logging of errors at each level has the additional benefit of providing a path of software execution that was invoked in detecting and handling the error condition. This can be very useful for a software engineer in analyzing the error log and determining the cause of the error.

The level at which the error is finally handled (i.e., where the decision is made to either abort program execution or to take other drastic action) may vary significantly depending upon the type of software detecting the error. As a rule of thumb, errors will be handled at the lowest possible level at which the program can properly respond to the error.

A7.4 Conventions

The following will be used as the standard convention for calling a function and returning status and error information.

```
status = f_<pkg_a>_func (parm1, parm2,,,<error_code>)
```

The "status" variable will indicate whether the operation was a success or failure (i.e., can only take on a value of "c_spd_success" or "c_spd_failure") and the optional "error_code" will provide specific error code information in the case where the function was a failure.

The definition of possible values for the "error_code" variable will be defined in the interface file for the "pkg_a" package. The error code mnemonic should be descriptive enough so that when it is used in the code, it is apparent what the error is.

A7.5 Logging

In order to have commonality of messages logged to an error log file (whether it's an error log on the Testbed or DSP), the following error line format is used:

```
<time-stamp> <function-name> <severity-code> < message-text>
```

"time-stamp": indicates the time at which the error message was output (i.e., 12:10:15). Note, this is not provided for error messages logged by the DSP as there is no clock.

"function-name":	identifies the name of the function logging the error message (i.e., "f_crad_validate_alt")
"severity-code":	indicates the severity of the error message (one of "FATAL", "WARN" and "INFO")
"message-text":	provides the complete error message text clearly identifying the condition at fault (i.e., "invalid altitude: -3000")

Normally this will fit on an 80 character line. If the <message-text> field is too long, it will be wrapped onto the next line.

A7.6 CERR Package

This package will be used by the Controller software to format and log messages to an error log, see Appendix D1.18 for design details.

A7.7 Error Log Files

Given the SPD operational concept and the current SPD design, there is no need to have one system wide error log file capturing all SPD error information. The SPD will instead provide two types of error logs:

- HMI error logs
- activity error logs

a) HMI error logs are generated by the Testbed Controller. The Controller provides the user with HMI functions in each of the following tasks:

  - Satellite/Radar Definition Task
  - Environment Definition Task
  - Simulation Run Definition Task
  - Generation Monitor/Control Task
  - Simulation Monitor/Control Task

  In each task, the user can select different parameter definitions to operate on and each task is independent of other HMI tasks. As such, the error information will be captured in an error log file associated with each of the above tasks. This will allow error information to be analyzed for each separate HMI activity.

b) Activity error logs are associated with the execution of a specific definition (i.e., satellite/radar and environment definition combination for signal generation; and simulation run definition for signal data playback and signal processing). As such, the error information will be captured in an error log associated with a generation or run identifier (see Table A-5 for file naming convention). This allows error information to be analyzed for each separate signal generation or simulation run activity.

For signal generation, the activity error log file generated will consist of two parts:

1) Controller error log information from monitoring RSS interface;

2) RSS error log information from performing the signal generation.

On completion of the signal generation activity, the Controller will append the RSS error log information (initially captured in a seperate RSS generation error log) to the Controller error log for the specified signal generation identifier.

For simulation run, the error log file generated will consist of four parts:

1) Controller error log information from monitoring the RSS and DSP interfaces;

2) RSS error log information from performing the signal data playback;

3) ESP error log information from ESP signal processing functions;

4) RSP error log information from RSP signal processing functions.

On completion of the simulation run activity, the Controller will append the RSS, ESP and RSP error log information (initially captured in separate RSS, ESP and RSP error logs) to the Controller error log for the specified simulation run identifier.

APPENDIX B

RSS CSC DESIGN PROCESS

By: M. Baskin (RCL)
R. Collyer (RCL)

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

Page

APPENDIX B RSS CSC DESIGN PROCESS B-1

B1 RCL Software Design Methodology B-1

B2 Software CSC Definition B-6

B2.1 Introduction B-6
B2.2 Task Definition By Requirements B-10
B2.3 Decomposition Analysis B-14
B2.4 Task Definition Changes B-14
B2.5 Interface Changes B-14

B3 Software Requirements Logic B-16

B3.1 Introduction B-16
B3.2 CDADL Output Listing B-16
B3.3 CDADL Requirements Logic Report B-18
B3.4 Signal Generation Controller (SGEN) B-21
B3.5 Initialize Run (SRUN) B-26
B3.6 Burst Initialization (SBIT) B-31
B3.7 Burse Modelling (SBMT) B-37
B3.8 Playback Manager (SPMT) B-46
B3.9 DSP Interface (TDIT) B-49
B3.10 DSP Output Task (TIOP) B-52

B4 Type Definitions and Global Data Dictionary B-55

B4.1 Introduction B-55

B5 Task Communications Design B-57

B5.1 Introduction B-57
B5.2 Interface Between the RSS SCS and External CSCs B-57
B5.3 CSC/CSU Interfaces B-62

B6 Design Error Handling Logic B-67

B6.1 Introduction B-67
B6.2 SunOS Errors 68 .... B-68
B6.3 CSPI Errors B-69
B6.4 VxWorks Errors B-69
B6.5 VASP Errors B-69
B6.6 RSS Application Errors B-69

B7 CSU Design B-70

B7.1 Signal Generation Manager CSU (SSGM) ... B-70
B7.2 Signal Generation Controller B-74
B7.3 Run Initialization CSU (SRUN) B-78
B7.4 Burst Initialization CSU (SBIT) B-82
B7.5 Burst Modelling CSU (SBMT) B-86
B7.6 Burst Modelling Test Plan B-89
B7.7 RSS Utilities CSU (SUTL) B-91
B7.8 Signal Playback Manager CSU (SPMT) B-93
B7.9 DSP Interface CSU (TDIT) B-97
B7.10 DSP Output CSU (TIOP) B-102
B7.11 RPC Server CSU (TRSV) B-107

FIGURES

		Page
B-1	RCL SOFTWARE DEVELOPMENT PROCESS	B-2
B-2	INTER-TASK COMMUNICATION SYMBOLS	B-9
B-3	INTER-TASK COMMUNICATION DIAGRAM	B-15
B-4	SIGNAL GENERATION MANAGER (SSGM) LOGIC FLOW	B-71
B-5	SIGNAL GENERATION CONTROLLER (SGEN) LOGIC FLOW	B-75
B-6	RUN INITIALIZATION (SRUN) LOGIC FLOW	B-79
B-7	BURST INITIALIZATION (SBIT) LOGIC FLOW	B-83
B-8	BURST MODELLING (SBMT) LOGIC FLOW	B-87
B-9	SIGNAL PLAYBACK MANAGER (SPMT) LOGIC FLOW DIAGRAM	B-93
B-10	DSP INTERFACE CSC/CSU CONTEXT DATA FLOW DIAGRAM	B-98
B-11	DSP INTERFACE CSC/CSU HIGH-LEVEL DATA FLOW DIAGRAM	B-99
B-12	TIOP CSC/CSU CONTEXT DATA FLOW DIAGRAM	B-103
B-13	TIOP CSC/CSU HIGH-LEVEL DATA FLOW DIAGRAM	B-104
B-14	TRSV CSC/CSU CONTEXT DATA FLOW DIAGRAM	B-107
B-15	TRSV CSC/CSU HIGH-LEVEL DATA FLOW DIAGRAM	B-108

TABLES

Page

B-1 RSS SOFTWARE DOCUMENTATION B-3
B-2 INTERTASK COMMUNICATION DEFINITIONS B-13
B-3 TESTBED CONTROLLER COMMANDS TO SIGNAL GENERATION MANAGER B-58
B-4 TESTBED CONTROLLER MESSAGES TO PLAYBACK MANAGER B-59
B-5 SIGNAL GENERATION MANAGER TO TESTBED CONTROLLER MESSAGES B-60
B-6 PLAYBACK MANAGER TO TESTBED CONTROLLER MESSAGES B-60
B-7 SIGNAL GENERATION MANAGER TO SIGNAL GENERATION CONTROLLER MESSAGES B-64
B-8 SIGNAL GENERATION CONTROLLER TO SIGNAL GENERATION MANAGER MESSAGES B-64
B-9 PLAYBACK MANAGER TO DSP INTERFACE MESSAGES B-66
B-10 SSGM MODULE TEST PLAN B-72
B-11 SGEN MODULE TEST PLAN B-76
B-12 SRUN MODULE TEST PLAN B-80
B-13 SBIT MODULE TEST PLAN B-84
B-14 SBMT MODULE TEST PLAN B-89
B-15 SUTL MODULE TEST PLAN B-91
B-16 SPMT MODULE TEST PLAN B-94
B-17 TDIT MODULE TEST PLAN B-100
B-18 TIOP MODULE TEST PLAN B-105
B-19 TRSV MODULE TEST PLAN B-109

APPENDIX B

RSS CSC DESIGN PROCESS

B1 RCL SOFTWARE DESIGN METHODOLOGY

Raytheon Canada Limited's Software Design Methodology is based on the life-cycle development methodology with DOD-STD-2167A milestones, reviews and deliverables defined by specific contracts. To manage the collection of technical information in an orderly manner which permits both the capture of the software development thought process and the assessment of technical development progress and maturity, Raytheon Canada Limited has devised a set of development activities for each phase of development. The activities defined for detailed design and their associated interdependencies are shown in Figure B-1. The association between activities and deliverables produced for the SBR RSS design is given in Table B-1. The Define CSCs activity , Model Inter-CSC Communication activity and the Estimate CSC Resource Requirements activity and their associated Appendix B Documentation were completed as part of RSS Preliminary Design and are included here for continuity.

The information contained in Section B2 was presented at the SBR RSS PDR and is included with the RSS Detailed Design documentation for continuity. Introductions and relevant background comments are provided in each section to identify the significance of the section documentation to the development process and to give the reader insight into the rational for design decisions.

The following paragraphs provide a brief overview of each activity. The order in which the activities are presented is indicative of the order in which the activities are performed

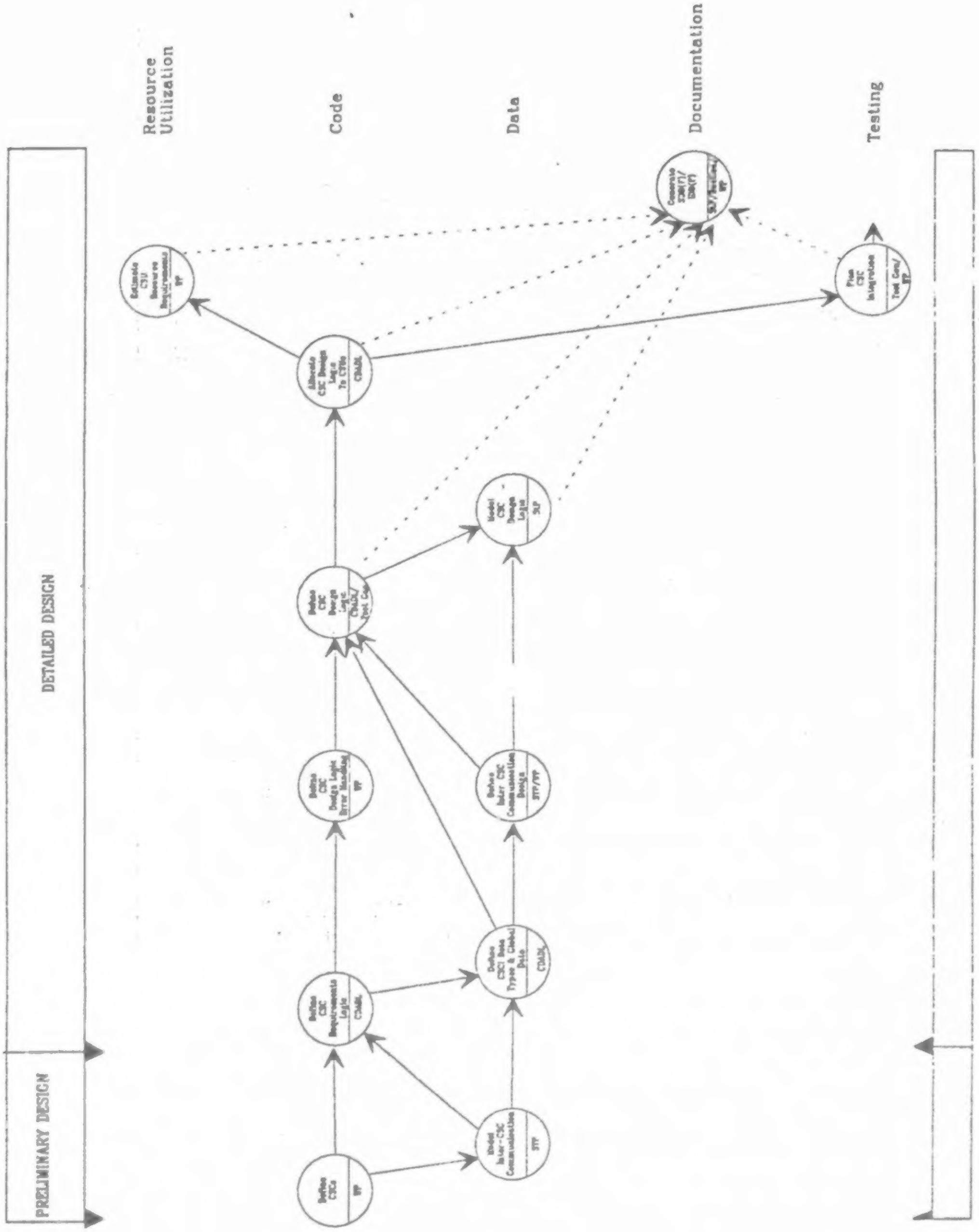

Resource Utilization
Code
Data
Documentation
Testing
PRELIMINARY DESIGN
DETAILED DESIGN
Define CSCs
PP
Model Inter-CSC Communication
STP
Define CSC Requirements Logic
CDADL
Define CSCI Data Types & Global Data
CDADL
Define CSC Design Logic Error Handling
PP
Define Inter CSC Communication Design
STP/PPP
Define CSC Design Logic
CDADL/ Tool Gen
Model CSC Design Logic
SLP
Allocate CSC Design Logic To CPUs
CDADL
Estimate CPU Resource Requirements
PP
Plan CSC Integration
Tool Gen/ PP
Generate SDD(T)/ IDD(T)
SLP/Tool Gen/ PP

Figure B-1 RCL SOFTWARE DEVELOPMENT PROCESS

Activity	**Appendix B Documentation**
Define CSCs	B2 Software CSC Definition
Model Inter-CSC Communication	B2 Software CSC Definition
Define CSC Requirements Logic	B3 Software Requirements Logic Definition
Define CSC Data Types and Global Data	B4 Type Defs and Global Data
Define CSC Communciation Design	B5 Task Communications Design
Define CSC Design Logic Error Handling	B6 Design Logic Error Handling
Define CSC Design Logic	B7 CSU Detailed Design
Model CSC Design Logic	B7 CSU Detailed Design
Allocate CSC Design Logic to CSU's	B7 CSU Detailed Design
Plan CSC Integration	B7 CSU Detailed Design
Define CSC/CSU Data	B7 CSU Detailed Design

Table B-1 RSS SOFTWARE DOCUMENTATION

during design. The activities Define CSC Design Logic, Model CSC Design Logic, Allocate CSC Design Logic to CSU's and Plan CSC Integration are repeated for each CSC.

The Define CSCs activity uses top level CSC functions and top level CSC data in conjunction with classic decomposition criteria (time critical functionality, time dependent functionality, functional cohesion, temporal cohesion) and software designer experience to identify unique low-level CSCs.

The Model Inter-CSC Communication activity involves selecting the appropriate classical process communication concepts (loosely coupled, closely coupled, event driven, information hiding) for inter-CSC communication based on inter-CSC temporal communication needs. The objective of this activity is to generate an Inter-CSC Communication Diagram.

The Define CSC Requirements Logic activity involves the generation of requirements logic written in structured English pseudo code. Requirements logic is an ordered interpretation of CSC requirements using system engineering defined language elements. The requirements logic is used to validate the correctness and completeness of application knowledge transfer from system engineering to design engineering prior to introducing target dependent and software dependent design criteria. This activity centres around the use of CDADL (C-based Design and Documentation Language) which permits the use of C control flow constructs and provides an automatic checking mechanism for correct control flow syntax.

The Estimate CSC Resource Requirements activity involves confirming the ability of the hardware resources to meet performance requirements. CSC resource requirements are based on CSC requirements logic pseudo code.

The Define CSC Data Types and Global Data activity involves "typing" CSC data elements found in the CSC requirements logic in the types and objects permitted by the application language and analysing the CSC requirements logic and inter-CSC communication to define the global data required for inter-CSC communication. This activity centres around the generation of include files (.h files) formatted with additional CDADL specific keywords and symbols which permit the automatic generation of data dictionary reports, object cross reference reports and type cross reference reports. Since the reports are produced from include files it follows that the reports can easily be reproduced as required during design and implementation thereby ensuring that the reports are both accurate and current.

The Define Inter-CSC Communication Design activity involves defining the appropriate inter-CSC communication mechanisms to match the inter-CSC communication concepts defined during the Model Inter-CSC Communication activity. The inter-CSC communication mechanisms are constrained by target hardware capabilities and dependent on available mechanisms provided by operating systems. The mechanisms defined in this activity are used to more completely define the CSC requirements logic statements that imply communication.

The Define CSC Design Logic Error Handling activity involves defining a consistent error handling philosophy to be applied across all CSCs. The CSC design logic error handling defines how to handle off-the-shelf function return errors (such as operating system call errors) and the extent to which defensive coding will be applied. Definition of CSC design logic error handling largely requires the extraction of information from existing vendor documentation. The error handling philosophy defined in this activity is used to more completely define the CSC design logic statements which require error handling such as operating system call design logic.

The Define CSC Design Logic activity involves adding software design logic (structured English pseudo code) to existing CSC requirements logic thereby launching the CSC into the chosen application hardware and software target environment. Software design logic includes inter-CSC communication logic, error handling logic, computation logic and defensive coding logic like checks for divide by zero. This activity centres around the use of CDADL which permits the extension of requirements logic produced during the Define CSC Requirements Logic activity and provides an automatic checking mechanism for correct control flow syntax.

The Allocate CSC Design Logic to CSU's activity involves the definition of CSU's and their modules, the allocation of CSC design logic (produced in the Define CSC Design Logic activity) to CSU modules and the identification of CSU module inputs and outputs. Similar to the Define Design Logic activity the use of CDADL provides the automatic checking mechanism to ensure that design logic maintains correct control flow syntax.

The Model CSC Design Logic activity involves creating a CSC level dataflow diagram from the CSC design logic and the inter-CSC communication diagram. This activity provides an alternate representation of the design logic which spans a visual-conceptual gap and evokes discussion thereby adding a pictorial dimension to CSC design logic scrutiny.

The Plan CSC Integration activity involves developing CSC integration test objectives consistent with allocated CSC functionality defined in the CSC design logic and stating CSU module testing intentions: desk inspection testing, functional testing and/or performance testing. All CSU modules are desk inspected by the CSC design engineer or a designated alternate. Functional and/or performance testing will be conducted on critical and/or complex CSU modules. CDADL's companion product TestGenC is used to enumerate path testing cases for complex CSU modules.

B2 SOFTWARE CSC DEFINITION

B2.1 Introduction

The process of defining the software tasks for a Computer Software Configuration Item (CSCI) or a top-level CSC consists of two phases.

The first phase is conducted while considering the System Requirements and Configuration Item Requirements. The CSCI is decomposed into tasks and the inter-task communication established based on requirements only.

The second phase is conducted when design and implementation is being considered. The objective is to produce the final task decomposition and inter-task communication that will be designed and implemented. The results of the first phase are evaluated and the evaluation results documented. If the task decomposition and inter-task communication by requirements is acceptable by design and implementation issues, the phase is complete. Otherwise the task decomposition and inter-task communication are altered in order to satisfy design and implementation.

The main consideration in decomposing a CSCI into concurrent tasks is the asynchronous nature of the functions within the system. The processes in the data flow diagram are analyzed to identify which may run concurrently and which are sequential in nature. By this means, tasks are identified. One process may correspond to one task, or one task may encompass several processes. The following criteria are used to decompose a CSCI into tasks during phase 1:

1. **Dependency on I/O**
   Depending on input or output, a process is often constrained to run at a speed dictated by the speed of the I/O device with which it is interacting. In this case, the process needs to be a separate task.

2. **Time-Critical Functions**
   A time-critical function needs to run at a high priority and therefore needs to be a separate task.

3. **Computational Requirements**
   A computationally intensive function (or set of functions) can run as a lower priority task consuming spare CPU cycles.

4. **Functional Cohesion**
   Processes that perform a set of closely related functions can be grouped together into a task. Since the data traffic between these functions may be high, having them as separate tasks will increase system overhead, whereas implementing each function as a separate module within the same task ensures functional cohesion both at the module and task levels.

5. **Temporal Cohesion**
   Certain processes perform functions that are carried out at the same time. These functions may be grouped into a task so that they are executed each time the task receives a stimulus.

   Although temporal cohesion is not considered a good module decomposition criterion in Structured Design, it is considered to be acceptable at the task level. Each function should be implemented as a separate module to achieve functional cohesion at the module level. These modules in turn are grouped into the task thereby achieving temporal cohesion at the task level.

6. **Periodic Execution**
   A process that needs to be executed periodically can be structured as a separate task that is activated at regular intervals.

After the decomposition of the CSCI into tasks, the interfaces between tasks must be defined. On the data flow diagram, the interfaces are in the form of data flows, event flows or data stores. However, at this time a more detailed definition of the task interfaces is required as any errors or inconsistencies in the design of the task interfaces may lead to deadlocks that, in order to be corrected, may require major design changes.

Thus, for each interface between two tasks, at least the interface type (communication or synchronization) and the method of communication (asynchronous or synchronous) should be specified in addition to the definition of the data passed through the interface.

Communication between tasks is classified into three categories:

a) **Loosely coupled message queue**:
   - one task needs to pass information to the other
   - both two tasks may proceed independently

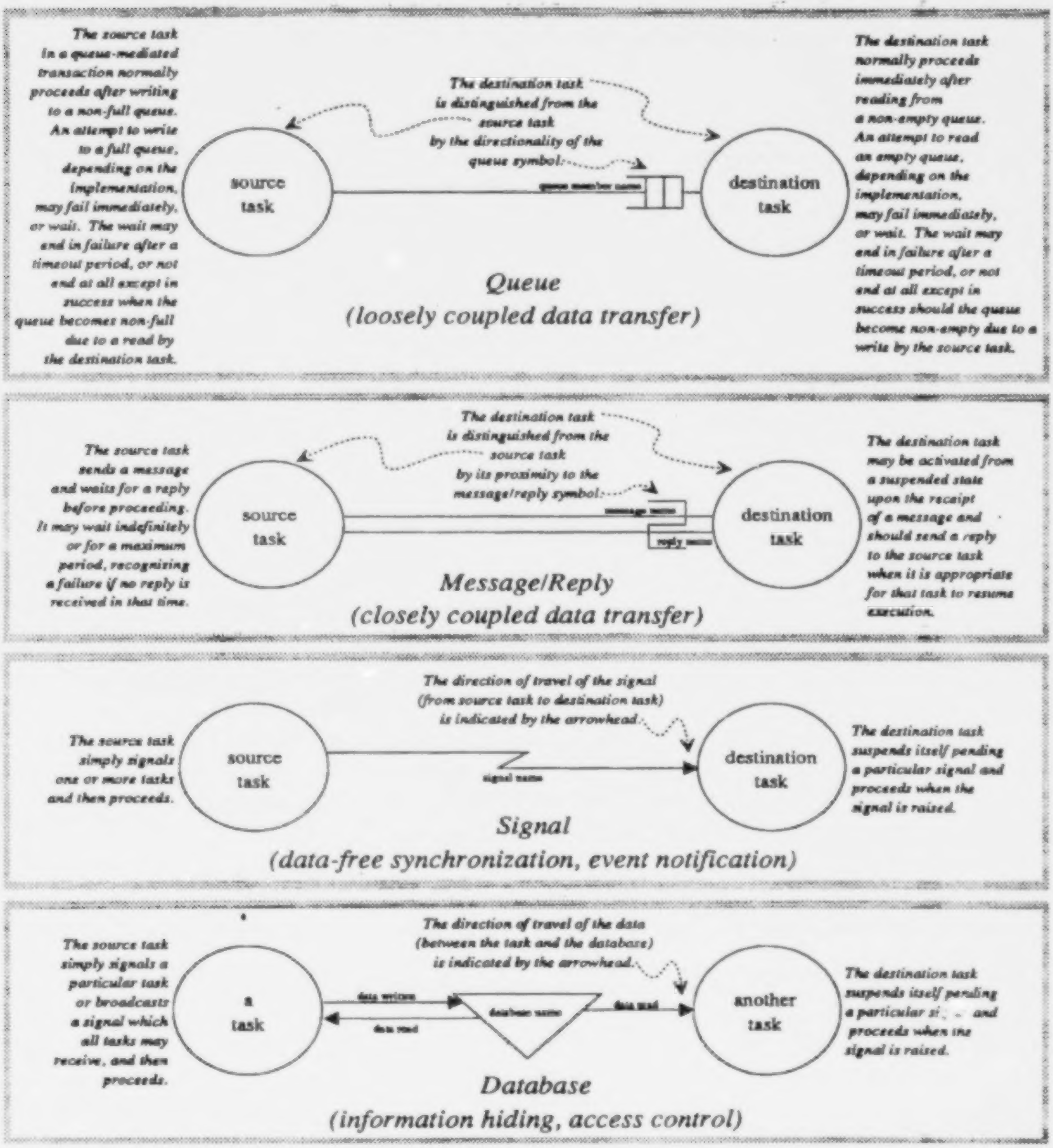
b) **Closely coupled message/reply**:
   - information is passed from one task to another
   - first task cannot proceed until it has received a reply from the second

c) **Event signal**:
   - notification of occurrence of event
   - no data transfer required

d) **Database**:
   - any number of tasks need to store and share information
   - controlled access necessary to maintain integrity of information

Communication between tasks is modelled using DARTS (Design Approach for Real Time Systems) inter-task modelling conventions. Figure B-2 defines the symbols used to depict inter-task communication.

Symbols Used To Depict Inter-Task Communications



Queue
(loosely coupled data transfer)

The source task in a queue-mediated transaction normally proceeds after writing to a non-full queue. An attempt to write to a full queue, depending on the implementation, may fail immediately, or wait. The wait may end in failure after a timeout period, or not end at all except in success when the queue becomes non-full due to a read by the destination task.

The destination task is distinguished from the source task by the directionality of the queue symbol.

The destination task normally proceeds immediately after reading from a non-empty queue. An attempt to read an empty queue, depending on the implementation, may fail immediately, or wait. The wait may end in failure after a timeout period, or not end at all except in success should the queue become non-empty due to a write by the source task.

Message/Reply
(closely coupled data transfer)

The source task sends a message and waits for a reply before proceeding. It may wait indefinitely or for a maximum period, recognizing a failure if no reply is received in that time.

The destination task is distinguished from the source task by its proximity to the message/reply symbol.

The destination task may be activated from a suspended state upon the receipt of a message and should send a reply to the source task when it is appropriate for that task to resume execution.

Signal
(data-free synchronization, event notification)

The source task simply signals one or more tasks and then proceeds.

The direction of travel of the signal (from source task to destination task) is indicated by the arrowhead.

The destination task suspends itself pending a particular signal and proceeds when the signal is raised.

Database
(information hiding, access control)

The source task simply signals a particular task or broadcasts a signal which all tasks may receive, and then proceeds.

The direction of travel of the data (between the task and the database) is indicated by the arrowhead.

The destination task suspends itself pending a particular signal and proceeds when the signal is raised.

Figure B-2 INTER-TASK COMMUNICATION SYMBOLS

B2.2 Task Definition By Requirements

B2.2.1 Task Definitions

Task definitions are described on a task by task basis. For each task, the functions for which it is responsible and the reasoning for the decomposition, are given. A summary of the dataflows with receiving and sending tasks is given in Table B-2.

B2.2.1.1 Signal Generation Manager Task (GM)

Functions

- Manage Signal Generation (1.1)
  Requirements Allocation
    - All "shalls"
- Initialize Run (1.2)
  Requirements Allocation
    - All "shalls"

Creation Rationale

- Functional Cohesion
  This task is responsible for control and management of signal generation.

Operational States of Execution

- Signal Generation

B2.2.1.2 Burst Initialization Task (BI)

Functions

- Initialize Burst Model (1.3)
  Requirements Allocation
    - All "shalls"

Creation Rationale

- Temporal Cohesion
  This task is responsible for all operations that must be performed before the next burst worth of data can be generated.
- Unusual computational requirements
  The operations within this task are compute-intensive, and will make use of

specialized hardware.

Operational States of Execution
- Signal Generation

B2.2.1.3 Burst Modelling Task (BM)

Functions
- Model Received Data (1.4)
    Requirements Allocation
    - All "shalls"

- Model Receiver Front End (1.5)
    Requirements Allocation
    - All "shalls"

Creation Rationale
- Temporal Cohesion
    This task encompasses all operations needed to generate one burst worth of data.

- Unusual computational requirements
    The operations within this task are compute-intensive, and will make use of specialized hardware.

Operational States of Execution
- Signal Generation

B2.2.1.4 Playback Manager Task (PM)

Functions
- Manage Radar Signal Playback (2.1)
    Requirements Allocation
    - All "shalls"

Creation Rationale
- Functional Cohesion
    This task encompasses all operations relating to the set-up and management of a playback run.

Operational States of Execution

- Signal Playback

B2.2.1.5 DSP Interface Task (DI)

Functions
- Provide DSP Interface (2.2)
  Requirements Allocation
  - All "shalls"

Creation Rationale
- Functional Cohesion
  This task encompasses all operations relating to the transmission of SRSD to the DSP.

- Dependency on I/O
  This task manages all I/O between the RSS and the DSP.

- Time-critical functionality
  Because of the real-time constraints for the signal playback, this task is located on specialized hardware.

Operational States of Execution
- Signal Playback

Dataflow	Reply Dataflow	Interface Category	Sender	Receiver	Execution State
Signal Generation Commands	N/A	Loosely coupled	Testbed Controller	Signal Gen Manager Task	Signal Generation
Signal Generation Parameters	N/A	Information Database	Testbed Controller	Signal Gen Manager Task	Signal Generation
RSS Responses from SG	N/A	Loosely coupled	Signal Generation Manager Task	Testbed Controller	Signal Generation
Signal Generation Status	N/A	Loosely coupled	Signal Generation Manager Task	Testbed Controller	Signal Generation
DSP Sig Proc Parameters	N/A	Information Database	Signal Gen Manager Task	Testbed Controller	Signal Generation
Generated SRSD	N/A	Information Database	Burst Modelling Task	Testbed Controller	Signal Generation
RSS Intermediate Results	N/A	Information Database	Burst Modelling Task	Testbed Controller	Signal Generation
Burst Initialization Parameters	N/A	Loosely coupled	Signal Gen Manager Task	Burst Initialization Task	Signal Generation
Return Status from BI	N/A	Loosely coupled	Burst Initialization Task	Signal Gen Manager Task	Signal Generation
Modelling Parameters	N/A	Loosely coupled	Signal Gen Manager Task	Burst Modelling Task	Signal Generation
Return Status from BM	N/A	Loosely coupled	Burst Modelling Task	Signal Gen Manager Task	Signal Generation
Front-end Parameters	N/A	Loosely coupled	Signal Gen Manager Task	Burst Modelling Task	Signal Generation
Burst Database	N/A	Loosely coupled	Burst Initialization Task	Burst Modelling Task	Signal Generation
Burst Updates	N/A	Loosely coupled	Burst Modelling Task	Burst Initialization Task	Signal Generation
Simulation Run Parameters	N/A	Information Database	Testbed Controller	Playback Manager Task	Signal Playback
Simulated Radar Signal Data	DSP Data Acknowledge	Closely coupled	DSP Interface Task	DSP	Signal Playback
Playback Commands	N/A	Loosely coupled	Controller Interface Task	Playback Manager Task	Signal Playback
Playback Status	N/A	Loosely coupled	Playback Manager Task	Controller Interface Task	Signal Playback
Pregenerated SRSD	N/A	Information Database	Testbed Controller	Playback Manager Task	Signal Playback
Buffered SRSD	N/A	Information Database	Playback Manager Task	DSP Interface Task	Signal Playback
RSS Responses from Playback	N/A	Loosely coupled	Playback Manager Task	Testbed Controller	Signal Playback
SRSD Commands	N/A	Loosely coupled	Playback Manager Task	DSP Interface Task	Signal Playback
Return Status from DI	N/A	Loosely coupled	DSP Interface Task	Playback Manager Task	Signal Playback

Table B-2 INTERTASK COMMUNICATION DEFINITIONS

B2.2.2 Summary

The results of task definition by requirements, comprising task decomposition and inter-task communication, is illustrated in Figure B-3.

The communication with external entities (peripheral equipment, external systems, etc.) is derived from the dataflow diagrams and is given to provide context and reference.

B2.3 Decomposition Analysis

B2.3.1 Introduction

This phase of software task design reconsiders the decomposition created from the functional requirements in the light of implementation constriants such as resource allocation and software development strategy. This may involve reevaluating the relative weights of the six criteria mentioned earlier in this document. It may also involve introducing new criteria, often based on the experience of the designer.

B2.3.2 Design and Implementation Decomposition Criteria

None.

B2.3.3 Analysis Conclusions

There was no further decomposition required.

B2.4 Task Definition Changes

No definitions changed.

B2.5 Interface Changes

No definitions changed.

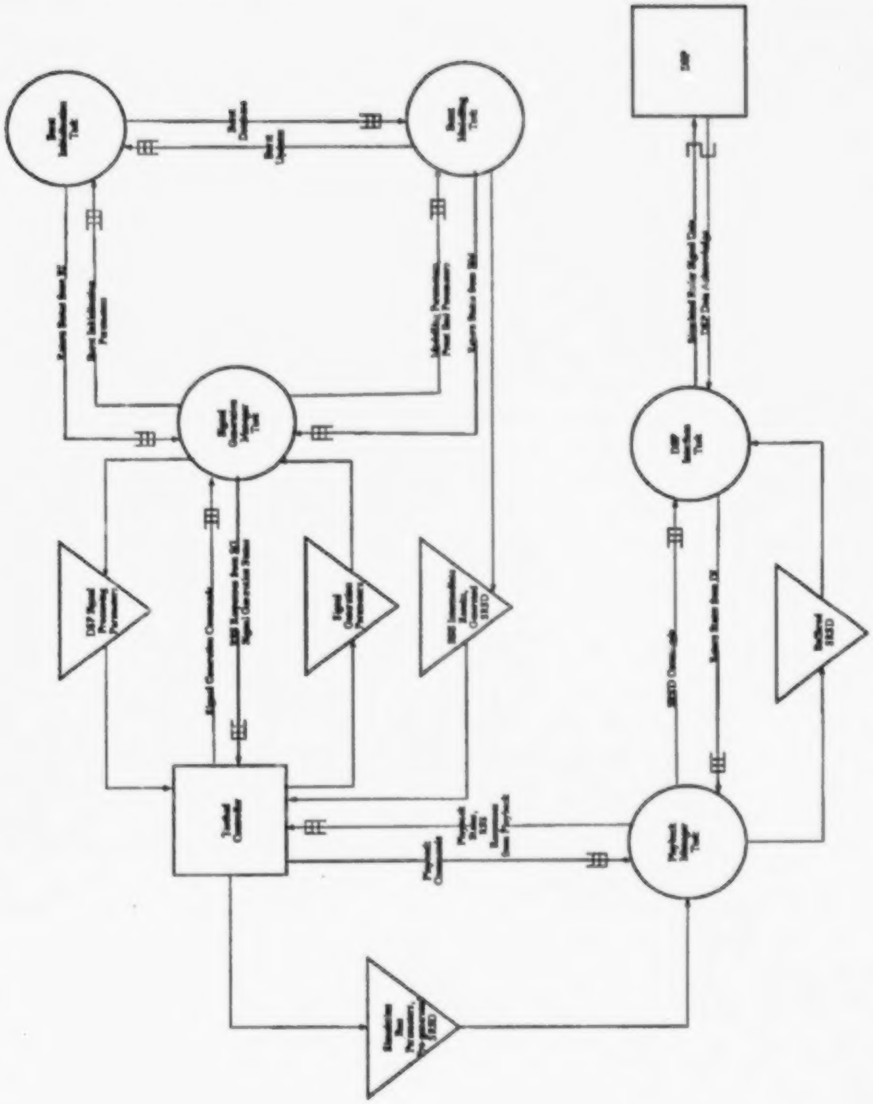


Figure B-3 INTER-TASK COMMUNICATION DIAGRAM

B3 SOFTWARE REQUIREMENTS LOGIC

B3.1 Introduction

This document describes the Requirements Logic for the CSCs of the RSS. The reports included in this document were generated from the CSC Requirements Logic source files using the C-based Design and Documentation Language (CDADL) processor. The intent of each report is described in section B3.2.

The software requirements logic is written in structured-English pseudocode and is an ordered interpretation of the CSC Development Specification requirements. The requirements logic serves the purpose of imposing structured logic upon the CSC requirements without imposing a design. The requirements logic pseudocode is organized by task as defined in the Software Task Definition design note and uses the CSC Development Specification data item names to provide continuity with the requirements defined in the CSC Development Specification. The requirements logic pseudocode is written at a high level to allow program system engineering to verify the designer's understanding and interpretation of the CSC requirements.

B3.2 CDADL Output Listing

B3.2.1 Introduction

The following sections describe the CDADL output listing included in this design note. The output listing begins with a CDADL Product banner page identifying the product name, the product version, the product serial number and the product authorized user.

B3.2.2 Requirements Logic

The requirements logic for each CSC source file given in the table of contents is presented indented according to the program structures. Nesting of control constructs is emphasized by vertical bars. Page and line numbers given in the report are used for cross-references in the remaining reports.

CSC global data item names used in the pseudocode are identifiable as concatenated words. For example, the CSC data item "SSP Longitude" appears in the pseudocode as "SSPLongitude". All global CSC data items referenced in the CSC Requirements Logic pseudocode are defined in the Type Defs and Global Data Dictionary design note.

For the sake of clarity of the algorithm, the pseudocode also references significant local variables which are not defined in the Global Data Dictionary. These items are referred to by name in lower case, for example "target range".

B3.2.3 TBD Report

The TBD Report is a report of all "To Be Defined" references in the CSC Requirements Logic. The TBD reference is used by the designer to flag problems encountered during requirements logic generation such as incorrect or incomplete requirements. The TBD Report cross-references the location of every TBD in a normal requirements logic statement and dumps the remainder of the comment after the cross-reference information.

B3.2.4 Error Summary Report

This report identifies any errors and warnings encountered during CDADL processing. CDADL ensures that the balancing of the flow control structures (ifs, for loops etc.) are correct.

B3.3 CDADL Requirements Logic Report

B3.3.1 Signal Generation Manager (SSGM)

```
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
[] [] [] []                                                          [] [] [] []
[] [] [] []                                                          [] [] [] []
[] [] [] []                                                          [] [] [] []
   [] []                                                                [] []
   [] []                         CDADL PROCESSOR                        [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                          VERSION 2.4.6                         [] []
   [] []                                                                [] []
   [] []                     SERIAL NUMBER: C38U0011                    [] []
   [] []                                                                [] []
   [] []                        AUTHORIZED USER:                        [] []
   [] []                        Raytheon - Canada                       [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
[] [] [] []                                                          [] [] [] []
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
```

```
/*%% Signal generation manager Task Requirements Logic */

#include "ssgm.h"

int ssgm ()   /*% Signal Generation Manager Task */

{

    /*+ ssgm requirements logic satisfies those functional requirements for the Signal Generation
    Manager Task relating to message communications with the testbed controller
    as specified in the Software Task Definition document

    NOTE: this task was separated out from sgen.c in order to satisfy the mechanics of RPC
    communication.  In abstract design terms, it is not needed; in implementation terms,
    it is  */

/*|

    initialize interface to testbed controller

    spawn sgen and initialize interface to it

    set RSS_state to "idle"

    DO
    |
    |   wait for incoming message from testbed or signal generation
    |
    |   SWITCH on incoming message
    |   |
    |   |   CASE StartGenerationCommand from testbed controller
    |   |   CASE AbortGenerationCommand from testbed controller
    |   |   |
    |   |   |   acknowledge the message
    |   |   |   send the message to sgen and wait for an acknowledge or timeout
    |   |   |   IF timeout THEN
    |   |   |   |   try to assess the state of sgen, and reestablish contact or restart it.  The
    |   |   |   |   subtlety or lack thereof with which this is done is TBD
    |   |   |   |   IF sgen remains incommunicado THEN
    |   |   |   |   |   set RSS_state to "hung"
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   <-------break
    |   |   |
    |   |   CASE StatusMessage from testbed controller
    |   |   |
    |   |   |   acknowledge the message, giving RSS_state
    |   <-------break
    |   |   |
    |   |   CASE PingMessage from testbed controller
    |   |   |   acknowledge the message
    |   <-------break
    |   |   |
    |   |   CASE ShutdownGenerationCommand from testbed controller
    |   |   |
    |   |   |   acknowledge the message
    |   |   |   send the message to sgen
    |   |   |   exit
    |   |   |
    |   |   CASE of StartGenerationResponse from sgen
    |   |   CASE of AbortGenerationResponse from sgen
    |   |   |
    |   |   |   send the message to the testbed controller
    |   |   |   wait for an acknowledgement
    |   <-------break
    |   |   |
    |   ENDSWITCH
    |
    DOWHILE forever
    ENDDO

*/
73
74     }
```

```
                                                                                 PAGE 3
                                              Requirements Traceability
================================================================================
SUBJECT
    PAGE NO.   LINE NO.      ENCLOSING FUNCTION

   NO MARKS TO REFERENCE
                                                                                   PAGE
                                                   ERROR SUMMARY
================================================================================
PAGE NO.   LINE NO.    ERROR TEXT

   NO ERRORS TO REPORT
```

B3.4 Signal Generation Controller (SGEN)

```
[][][][][][][][][][][][][][][][][][][][][][][][][][][]
[][][][]                                      [][][][]
[][][][]                                      [][][][]
[][][][]                                      [][][][]
  [][]                                          [][]
  [][]             CDADL PROCESSOR              [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]              VERSION 2.4.6               [][]
  [][]                                          [][]
  [][]         SERIAL NUMBER: C38U0011          [][]
  [][]                                          [][]
  [][]             AUTHORIZED USER:             [][]
  [][]            Raytheon - Canada             [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]                                          [][]
[][][][]                                      [][][][]
[][][][][][][][][][][][][][][][][][][][][][][][][][][]
```

LINE sgen.c PAGE

```
/*%% Signal generation controller Task Requirements Logic */

#include "sgen.h"

int sgen ()   /*% Signal Generation Controller Task */

{

    /*+ sgen requirements logic satisfies those functional requirements for the Signal Generation
    Manager Task relating to message communications with the testbed controller
    as specified in the Software Task Definition document */

/*|

    initialize interface to ssgm

    set RunState to "idle"

    DO
    |
    |   check for an incoming message from ssgm
    |
    |   IF a message is present THEN
    |   |   acknowledge the message
    |   |
    |   |   SWITCH on incoming message type
    |   |   |
    |   |   |   CASE of StartGenerationCommand
    |   |   |   |
    |   |   |   |   IF RSS_State is "idle" THEN
    |   |   |   |   |   set RSS_State to "running"
    |   |   |   |   |   set RunState to "running"
    |   |   |   |   |   initialize the error log
    |   |   |   |   |   send a StartingRun message to the error log
    |   |   |   |   |   activate srun
    |   |   |   |   |
    |   |   |   |   ELSE
    |   |   |   |   |   send a StartGenerationResponse message to ssgm with status
    |   |   |   |   |   "currently in progress"
    |   |   |   |   |   send a StartGenerationRequestedWithRunInProgress message to the error log
    |   |   |   |   ENDIF
    |   |   <-------break
    |   |   |   |
    |   |   |   CASE of AbortGenerationCommand
    |   |   |   |
    |   |   |   |   IF RSS_State is "idle" THEN
    |   |   |   |   |   set AbortStatus to "abort compl"
    |   |   |   |   |   send an AbortGenerationResponse message to ssgm
    |   |   |   |   |
    |   |   |   |   ELSE
    |   |   |   |   |   set RunState to "aborting"
    |   |   |   |   ENDIF
    |   |   <-------break
    |   |   |   |
    |   |   |   CASE of ShutdownGenerationCommand
    |   |   |   |   IF the RSS_State is "idle" and RunState is "idle" THEN
    |   |   |   |   |   exit
    |   |   |   |   |
    |   |   |   |   ELSE
    |   |   |   |   |   set RunState to "terminating"
    |   |   |   |   ENDIF
    |   |   <-------break
    |   |   |   |
    |   |   ENDSWITCH
    |   ENDIF
    |
    |   IF RSS_State is "running" THEN
    |   |
    |   |   check for a CompletionStatus from srun, sbit, sbmt
    |   |
    |   |   IF CompletionStatus available THEN
    |   |   |
|    |    |    SWITCH on issuer
|    |    |    |
|    |    |    |    CASE of srun
|    |    |    |    |
|    |    |    |    |    IF CompletionStatus is "successful" THEN
|    |    |    |    |    |
|    |    |    |    |    |    SWITCH on RunState
|    |    |    |    |    |    |
|    |    |    |    |    |    |    CASE "running"
|    |    |    |    |    |    |    |    activate sbit
|    |    |    |    |    |    <-------break
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    CASE "aborting"
|    |    |    |    |    |    |    |    clean up internal database
|    |    |    |    |    |    |    |    set RSS_State to "idle"
|    |    |    |    |    |    |    |    set RunState to "idle"
|    |    |    |    |    |    |    |    send a SignalGenerationAborted message to the error log
|    |    |    |    |    |    |    |    close the error log
|    |    |    |    |    |    |    |    set AbortStatus to "abort compl"
|    |    |    |    |    |    |    |    send an AbortGenerationResponse message to ssgm
|    |    |    |    |    |    <-------break
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    CASE "terminating"
|    |    |    |    |    |    |    |    clean up internal database
|    |    |    |    |    |    |    |    send a SignalGenerationTerminated message to the error log
|    |    |    |    |    |    |    |    close the error log
|    |    |    |    |    |    |    |    exit
|    |    |    |    |    |    |    |
|    |    |    |    |    |    ENDSWITCH
|    |    |    |    |    |
|    |    |    |    |    ELSE
|    |    |    |    |    |    clean up internal database
|    |    |    |    |    |    set RSS_State to "idle"
|    |    |    |    |    |    set RunState to "idle"
|    |    |    |    |    |    close the error log
|    |    |    |    |    |    send a StartGenerationResponse message to ssgm with
|    |    |    |    |    |    Status "terminated with error"
|    |    |    |    |    ENDIF
|    |    |    <-------break
|    |    |    |    |
|    |    |    |    CASE of sbit
|    |    |    |    |
|    |    |    |    |    IF CompletionStatus is "successful" THEN
|    |    |    |    |    |
|    |    |    |    |    |    SWITCH on RunState
|    |    |    |    |    |    |
|    |    |    |    |    |    |    CASE "running"
|    |    |    |    |    |    |    |    activate sbmt
|    |    |    |    |    |    <-------break
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    CASE "aborting"
|    |    |    |    |    |    |    |    clean up internal database
|    |    |    |    |    |    |    |    set RSS_State to "idle"
|    |    |    |    |    |    |    |    set RunState to "idle"
|    |    |    |    |    |    |    |    send a SignalGenerationAborted message to the error log
|    |    |    |    |    |    |    |    close the error log
|    |    |    |    |    |    |    |    set AbortStatus to "abort compl"
|    |    |    |    |    |    |    |    send an AbortGenerationResponse message to ssgm
|    |    |    |    |    |    <-------break
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    CASE "terminating"
|    |    |    |    |    |    |    |    clean up internal database
|    |    |    |    |    |    |    |    send a SignalGenerationTerminated message to the error log
|    |    |    |    |    |    |    |    exit
|    |    |    |    |    |    |    |
|    |    |    |    |    |    ENDSWITCH
|    |    |    |    |    |
|    |    |    |    |    ELSE
|    |    |    |    |    |    clean up internal database
|    |    |    |    |    |    set RSS_State to "idle"
|    |    |    |    |    |    set RunState to "idle"
|    |    |    |    |    |    close the error log
|    |    |    |    |    |    send a StartGenerationResponse message to ssgm with
|    |    |    |    |    |    Status "terminated with error"
```

```
    |   |   |   |   |   ENDIF
    |   |   |   <-------break
    |   |   |   |   |
    |   |   |   |   CASE of sbmt
    |   |   |   |   |
    |   |   |   |   |   IF CompletionStatus is "successful" THEN
    |   |   |   |   |   |
    |   |   |   |   |   |   compute what percentage of the generation run is now complete
    |   |   |   |   |   |   |
    |   |   |   |   |   |   SWITCH on RunState
    |   |   |   |   |   |   |
    |   |   |   |   |   |   |   CASE "running"
    |   |   |   |   |   |   |   |   IF the run is finished THEN
    |   |   |   |   |   |   |   |   |   send SignalGenerationComplete to the error log
    |   |   |   |   |   |   |   |   |   set RSS_State to "idle"
    |   |   |   |   |   |   |   |   |   set RunState to "idle"
    |   |   |   |   |   |   |   |   |   close the error log
    |   |   |   |   |   |   |   |   |   send a StartGenerationResponse with "completed normally" stat
    |   |   |   |   |   |   |   |   |   to ssgm
    |   |   |   |   |   |   |   |   ELSE
    |   |   |   |   |   |   |   |   |   activate sbit
    |   |   |   |   |   |   |   |   |   IF PercentComplete has changed 5% since the last status
    |   |   |   |   |   |   |   |   |   message was sent to ssgm THEN
    |   |   |   |   |   |   |   |   |   |   send a StartGenerationResponse with "in progress" status
    |   |   |   |   |   |   |   |   |   |   giving the new percentage
    |   |   |   |   |   |   |   |   |   ENDIF
    |   |   |   |   |   |   |   |   ENDIF
    |   |   |   |   |   |   <-------break
    |   |   |   |   |   |   |   |
    |   |   |   |   |   |   |   CASE "aborting"
    |   |   |   |   |   |   |   |   clean up internal database
    |   |   |   |   |   |   |   |   set RSS_State to "idle"
    |   |   |   |   |   |   |   |   set RunState to "idle"
    |   |   |   |   |   |   |   |   send a SignalGenerationAborted message to the error log
    |   |   |   |   |   |   |   |   close the error log
    |   |   |   |   |   |   |   |   set AbortStatus to "abort compl"
    |   |   |   |   |   |   |   |   send an AbortGenerationResponse message to ssgm
    |   |   |   |   |   |   <-------break
    |   |   |   |   |   |   |   |
    |   |   |   |   |   |   |   CASE "terminating"
    |   |   |   |   |   |   |   |   clean up internal database
    |   |   |   |   |   |   |   |   send a SignalGenerationTerminated message to the error log
    |   |   |   |   |   |   |   |   exit
    |   |   |   |   |   |   |   |
    |   |   |   |   |   |   ENDSWITCH
    |   |   |   |   |   |
    |   |   |   |   |   ELSE
    |   |   |   |   |   |   clean up internal database
    |   |   |   |   |   |   set RSS_State to "idle"
    |   |   |   |   |   |   set RunState to "idle"
    |   |   |   |   |   |   close the error log
    |   |   |   |   |   |   send a StartGenerationResponse message to ssgm with
    |   |   |   |   |   |   Status "terminated with error"
    |   |   |   |   |   ENDIF
    |   |   |   <-------break
    |   |   |   |   |
    |   |   |   ENDSWITCH
    |   |   ENDIF
    |   ENDIF
    |
    DOWHILE forever
    ENDDO

*/

}
```

```
                                                                          PAGE 5
                                         Requirements Traceability

================================================================================
SUBJECT
   PAGE NO.   LINE NO.     ENCLOSING FUNCTION


  NO MARKS TO REFERENCE

                                                                        PAGE 6
                                              ERROR SUMMARY

================================================================================
PAGE NO.   LINE NO.    ERROR TEXT

  NO ERRORS TO REPORT
```

B3.5 Initialize Run (SRUN)

```
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
[] [] [] []                                                          [] [] [] []
[] [] [] []                                                          [] [] [] []
[] [] [] []                                                          [] [] [] []
   [] []                                                                [] []
   [] []                   CDADL PROCESSOR                              [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                    VERSION 2.4.6                               [] []
   [] []                                                                [] []
   [] []             SERIAL NUMBER: C38U0011                            [] []
   [] []                                                                [] []
   [] []                   AUTHORIZED USER:                             [] []
   [] []                 Raytheon - Canada                              [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
[] [] [] []                                                          [] [] [] []
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
```

srun.c

LINE

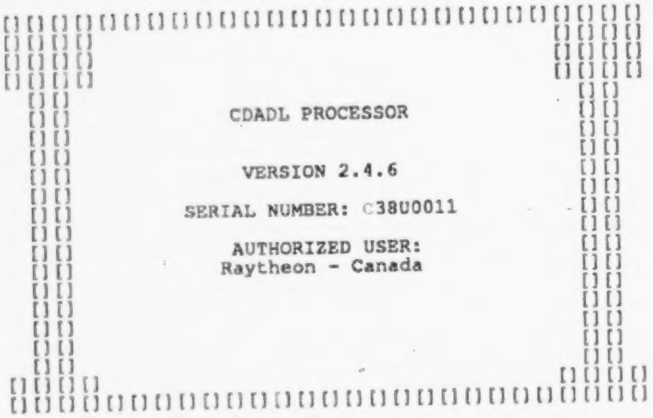
```
/*%% Signal generation initialization Task Requirements Logic */

#include "srun.h"

int srun ()   /*% Initialize Generation Run Task */

{

    /*+ srun is a child task of sgen, and satisfies a subset of the requirements allocated to sgen */

/*|

    set CompletionStatus to "successful"

    NOTE: This flag will be set to "failure" and control returned to sgen whenever an error
    is detected with any of the computations, filterings, file creations, etc.  For
    legibility, these checks are not shown at this level of pseudocode.

    ##### Setup Working Parameters #####
    ##### ------------------------- #####

    get Sat/antPathname, EnvPathname, OutputPathname and GenerationIdentifier from
    the incoming message

    create the DSPSignalProcessingParameters database, using OutputPathname, GenerationIdentifier

    load SignalGenerationParameters from the files pointed to by Sat/antPathname and
    EnvPathname

    Filter all SignalGenerationParameters into the required units (degrees to radians, metres to km,
    etc.)

    Set CurrentDwell, CurrentNCI, CurrentCPI to the first dwell, NCI, CPI

    Compute total NumberOfCPIs in the run

    FOR each of the intermediate products streams
    LOOP
    |   IF the intermediate product is selected THEN
    |   |   create the corresponding product file
    |   |   initialize its header block with Sat/antPathname, EnvPathname
    |   ENDIF
    ENDLOOP by product stream

    IF the InitialSSP is specified in beam centre parameters THEN
    |   Compute the SSPLatitude and SSPLongitude from the beam centre lat/long,
    |   and the beam centre azimuth and grazing angles
    |   Set these values back into Initial SSP, so they'll be retained through the run
    ELSE
    |   Set SSPLatitude and SSPLongitude to the values in InitialSSP
    ENDIF

    Set CurrentDirection to SatelliteDirection
    Compute OrbitRadius using Altitude
    Compute RotationalVelocity using OrbitRadius
    Compute K1 using OrbitRadius
    Compute K3 using RotationalVelocity, Inclination
    Compute Velocity from RotationalVelocity, OrbitRadius
    Compute NumberOfAntennas from NumberOfDPCAGroups and NumberOfAuxiliaryAntennas
    Compute NumberOfChannels from NumberOfAntennas and NumberOfDelayTaps
    Compute RangeQuant from SamplingRate, RangeQuantizationFactor
    Compute SlantRange from Altitude
    Compute NumRS from SlantRange, Altitude, RangeQuant
    Compute EquiRangeVector from NumRS, RangeQuant, SlantRange, Altitude
    Compute ReplicaInterpolationFactor  (TBD: may be done once per dwell)
    Initialize TransStartTime, TransEndTime, TransStartTimeRef, TransEndTimeRef
    Compute ElevationAngles from SlantRange, RangeQuant, NumRS, Altitude
    Compute Iratio from Oversampling factors
    Compute OutOff from Iratio

    IF the NoiseSeed is RSS-specified THEN
    |   generate NoiseSeedInteger
```

```
ELSE
|   initialize NoiseSeedInteger with the value in SignalGenerationParameters
ENDIF

IF the ScintillationSeed is RSS-specified THEN
|   generate a ScintillationSeedInteger
ELSE
|   initialize ScintillationSeedInteger with the value in SignalGenerationParameters
ENDIF

IF the EarthClutterSeed is RSS-specified THEN
|   generate an EarthClutterSeedInteger
ELSE
|   initialize EarthClutterSeedInteger with the value in SignalGenerationParameters
ENDIF

IF the DiscreteClutterSeed is RSS-specified THEN
|   generate a DiscreteClutterSeedInteger
ELSE
|   initialize DiscreteClutterSeedInteger with the value in SignalGenerationParameters
ENDIF

IF the VolumetricClutterSeed is RSS-specified THEN
|   generate a VolumetricClutterSeedInteger
ELSE
|   initialize VolumetricSeedInteger with the value in SignalGenerationParameters
ENDIF

FOR each of n jammers
LOOP
|   IF its JammerSeed is RSS-specified THEN
|   |   generate a JammerSeedInteger for sequence n
|   ELSE
|   |   initialize JammerSeedInteger with the value in SignalGenerationParameters
|   ENDIF
ENDLOOP by jammer

FOR each of m targets
LOOP
|   IF its TargetSeed is RSS-specified THEN
|   |   generate a TargetSeedInteger for sequence m
|   ELSE
|   |   initialize TargetSeedInteger with the value in SignalGenerationParameters
|   ENDIF
ENDLOOP by target

Compute NoiseWindow vector

Compute FirstChannel, the array specifying the first channel number associated with each
antenna

Generate NonlinearityParams, the table of nonlinearity parameters for the antennas

Compute Offset, the first sample in the output segment to use during decimation

Compute antenna random sequence lengths and generate RandU and RandV, the Gaussian
white noise samples for the antenna patterns

##### Compute Orbital Model Coefficients #####
##### ---------------------------------- #####

IF Inclination = 0 THEN
|   Compute TimeZero, OmegaZero for an equatorial orbit
|   using SSPLongitude, RotationalVelocity
|
ELSEIF Inclination = PI/2 THEN
|   Compute TimeZero, OmegaZero for a polar orbit
|   using SSPLatitude, SSPLongitude, CurrentDirection, RotationVelocity
|
ELSE
|   Compute TimeZero, OmegaZero for the general case
|   using SSPLatitude, SSPLongitude, Inclination, CurrentDirection
|
ENDIF
```

```

    ##### Compute Tropospheric Refraction Database #####
    ##### ---------------------------------------- #####

    IF LowerAtmosphere is "ON" THEN
    |   load the correct portion of TropRefDB, the refraction database, based on Humidity
    ENDIF

    ##### Compute Receiver Front End Database #####
    ##### ----------------------------------- #####

    Compute IFFilterResponse matrix

    FOR each channel
    LOOP
    |   IF CaptureIFFilterResponse is "yes" THEN
    |   |   store the IFFilterResponse for the channel in the intermediate product file
    |   ENDIF
    ENDLOOP by channel

    Compute OmegaIf, the A/D IF angular frequency

    FOR each channel
    LOOP
    |   Compute IQParameters
    |   Compute ADParameters
    ENDLOOP by channel

    ##### Compute Earth Clutter Map #####
    ##### ------------------------- #####

    FOR each valid clutter patch in the HMI
    LOOP
    |   IF the patch straddles the mercator edge THEN
    |   |   divide it into separate patches bounded by the edge
    |   ENDIF
    |
    |   FOR each patch (NOTE: this is for the broken-up subpatches or a single patch)
    |   LOOP
    |   |   Compute patch latitude limits
    |   |   FOR each latitude line within the limits
    |   |   LOOP
    |   |   |   Increment Npplat, the number of patch borders along the latitude line
    |   |   |   Compute the longitude extent
    |   |   |   Record patch information in ClutterMapDb
    |   |   ENDLOOP by latitude line
    |   ENDLOOP by patch
    ENDLOOP by valid patch

    optimize ClutterMapDb

    close DSPProcessingParameters

<---return CompletionStatus to sgen

*/

}
```

```
                                                                                PA
                                            Requirements Traceability

=========================================================================================
SUBJECT
    PAGE NO.   LINE NO.      ENCLOSING FUNCTION


   NO MARKS TO REFERENCE

                                                                             PAGE 6
                                                ERROR SUMMARY

=========================================================================================
PAGE NO.   LINE NO.    ERROR TEXT

   NO ERRORS TO REPORT
```

B3.6 Burst Initialization (SBIT)



```
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
[] [] [] []                                                             [] [] [] []
[] [] [] []                                                             [] [] [] []
[] [] [] []                                                             [] [] [] []
   [] []                                                                   [] []
   [] []                         CDADL PROCESSOR                           [] []
   [] []                                                                   [] []
   [] []                                                                   [] []
   [] []                          VERSION 2.4.6                            [] []
   [] []                                                                   [] []
   [] []                     SERIAL NUMBER: C38U0011                       [] []
   [] []                                                                   [] []
   [] []                         AUTHORIZED USER:                          [] []
   [] []                        Raytheon - Canada                          [] []
   [] []                                                                   [] []
   [] []                                                                   [] []
   [] []                                                                   [] []
   [] []                                                                   [] []
   [] []                                                                   [] []
   [] []                                                                   [] []
[] [] [] []                                                             [] [] [] []
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
```

```
/*%% Burst Initialization Task Requirements Logic */

#include "sbit.h"

int sbit ()   /*% burst initialization task  */

{

    /*+ sbit requirements logic satisfies all functional requirements for the Burst Initialization Tas
    as specified in the Software Task Definition document */
/*|

    set CompletionStatus to "successful"

    NOTE: This flag will be set to "failure" and control returned to sgen whenever an error
    is detected with any of the computations, filterings, file creations, etc.  For
    legibility, these checks are not shown at this level of pseudocode.

    Update CurrentCPI

    IF starting a new dwell THEN
    |   IF the beam pointing is specified in azimuth/grazing angles
    |   or boresight offset angles THEN
    |   |   Compute the BeamCentreLatitude and BeamCentreLongitude
    |   ELSE
    |   |   Set BeamCentreLatitude, BeamCentreLongitude to the values in BeamCentreLocation
    |   ENDIF
    ENDIF

    Compute SatelliteDirection at beginning of burst
    Compute SSPLatitude at beginning of burst
    Compute SSPLongitude at beginning of burst
    Compute SBRHeading using Inclination, SSPLatitude, CurrentDirection
    Compute K2 using SBRHeading, SSPLatitude

    ##### Compute Waveform Coefficients #####
    ##### ----------------------------- #####

    Compute RadarWavelength using RadarFrequency
    Compute TransmitStartTime
    Compute ReferenceRange, BoresightDepressionAngle, BoresightAzimuthAngle
    Compute RadialEarthRotation, RadialPlatformMotion

    IF starting a new dwell THEN
    |   Set AdjustedPRF to the value in the SignalGenerationParameters for the current dwell
    |   IF PRFAdjustment is "on" THEN
    |   |   Compute AdjustedPRF
    |   ENDIF
    ENDIF

    IF starting a new NCI THEN
    |   Compute CoherentBurstTime using PulsesPerCPI, AdjustedPRF
    ENDIF

    Compute DopQuant using CoherentBurstTime, DopplerQuantizationFactor, RadarWavelength
    Compute TransmitEndTime

    FOR each DPCA group
    LOOP
    |   FOR each phase centre in group
    |   LOOP
    |   |   Compute transmit-to-receive times for start and end of burst
    |   |   IF leading phase centre THEN
    |   |   |   Compute PosSamplingOffset, BurstStLeadingPC, NumberOfSamples, NumberOfJammerSamples,
    |   |   |   NumberOfClutterSamples, ReceiveStartTimeRef, ReceiveEndTimeRef,
    |   |   |   absolute-burst-receive-start-time
    |   |   ENDIF
    |   |   Initialize inter-phase-centre-time-delays
    |   ENDLOOP by phase centre
    |   Compute inter-phase-centre-time-delays
    ENDLOOP by DPCA group

```

```
IF starting a new dwell THEN
|   Compute PulseCompressionReplica
|   Compute DecimatedPulseReplica
|   Store DSPProcParamsDwell in DSPProcessingParameters file
|   initialize BurstStLeadingPC
|   set ReceiveReferenceTime to absolute-burst-receive-start-time
ENDIF

IF starting a new NCI THEN
|   Compute InterPhaseCentreSeparation
|   store DSPProcParamsNCI in DSPProcessingParameters file
ENDIF

store DSPProcParamsCPI in DSPProcessingParameters file

##### Compute Attenuation #####
##### ------------------- #####

FOR each precipitation cell defined in the HMI
LOOP
|   IF the cell is within the current field of view THEN
|   |   Compute PrecipDbase parameters for the cell
|   ENDIF
ENDLOOP

FOR grazing angles from 1 to 90
LOOP
|   Compute AtAttenuation for the grazing angle
ENDLOOP

##### Compute Range-Doppler Map #####
##### ------------------------- #####

Compute AngleOfMaximumDoppler

Compute NumIso, NumberOfIsodopElements

Compute MaximumIsodopVector

FOR each range swath
LOOP
|
|   Compute number-of-patches in range swath using MaximumIsodopVector, DopQuant
|   Compute number-of-type-5-patches in range swath using MaximumIsodopVector, DopQuant
|   Update the number-of-type-5-elements for each applicable isodop in CPC
|   Compute the midswath-elevation-angle
|   Compute the MidswathGrazingAngle and store in Graz
|   Compute MidswathRange and store in RangeVector
|   Compute midswath-range-squared
|
|   FOR each type-5 patch
|   LOOP
|   |   Compute area of type 5 patch and store in CPA
|   ENDLOOP
|
|   FOR each non-type-5 patch
|   LOOP
|   |   Determine patch type and parameters required for area calculation
|   |   Update the number of type j patches in CPC
|   |   Compute area of patch and store in CPA
|   ENDLOOP
|
ENDLOOP

##### Compute Volumetric Clutter Database #####
##### ----------------------------------- #####

FOR each precipitation cell within the current field of view
LOOP
|   Compute the cell geometry, cell representation and bounding equi-ranges
|   Compute reflectivity and path length through cell
|   FOR each intercepted range swath
|   LOOP
|   |   derive the swath description
|   |   FOR each quadrant representation
|   |   LOOP
|   |   |   IF the cell extends into or is with the quadrant THEN
|   |   |   |   derive the isodop extents
|   |   |   |   FOR each doppler swath
|   |   |   |   LOOP
|   |   |   |   |   Determine the number of scatterers in the isodop and the patch type
|   |   |   |   |   Compute PrecipDB parameters (element location, RCS)
|   |   |   |   ENDLOOP for doppler swath
|   |   |   ENDIF
|   |   ENDLOOP for quadrant representation
|   ENDLOOP for intercepted range swath
|
|   Compute Frollof, Fpeak, AtmosAtten
ENDLOOP for precipitation cell

FOR each chaff bundle within the current field of view
LOOP
|   IF the chaff bundle effects the current frequency THEN
|   |   Increment Nchaffs, the number of chaff bundles affecting the current CPI
|   |   Compute Chaffloc, Fcrolloff, Fcpeak
|   |   IF cell is inside the atmosphere THEN
|   |   |   Compute ChafAtmosAtten
|   |   ELSE
|   |   |   ChafAtmosAtten = 1
|   |   ENDIF
|   |   Compute ChaffRCS
|   ENDIF
ENDLOOP by chaff bundles


##### Compute Target Database #####
##### ----------------------- #####

FOR each target
LOOP
|   Set TargetValidity (yes/no)
|   IF TargetValidity is "yes" THEN
|   |   Compute TargetRange, TargetAttenuation
|   ENDIF
ENDLOOP by target

##### Compute Jammer Database #####
##### ----------------------- #####

FOR each jammer
LOOP
|   Set JammerValidity (yes/no)
|   IF JammerValidity is "yes" THEN
|   |   Compute JammerRange, JammerAttenuation
|   ENDIF
ENDLOOP by jammer


##### Compute Antenna Patterns #####
##### ------------------------ #####

IF the main antenna is CFA THEN
|   determine scanning parameters from Attitude, BeamLocation, SSPLatitude, SSPLongitude,
|   Altitude
|   IF the patterns are RSS-generated THEN
|   |   generate TxAntennaPattern
|   ELSE
|   |   load user-specified TxAntennaPattern
|   ENDIF
|   FOR each DPCA group
|   LOOP
|   |   FOR each phase centre within the DPCA group
|   |   LOOP
|   |   |   IF the patterns are RSS-generated THEN
|   |   |   |   generate RxAntennaPattern
|   |   |   ELSE
|   |   |   |   load user-specified RxAntennaPattern
|   |   |   ENDIF
|   |   ENDLOOP by phase centre
|   ENDLOOP by DPCA group
ENDIF

initialize sample start times for auxiliaries

IF the main antenna is RRA THEN
|   IF the patterns are RSS-generated THEN
|   |   generate TxAntennaPattern
|   |   generate RxAntennaPattern
|   ELSE
|   |   load user-specified TxAntennaPattern
|   |   load user-specified RxAntennaPattern
|   ENDIF
ENDIF

FOR each auxiliary antenna
LOOP
|   IF auxiliary type is single element THEN
|   |   generate RxAntennaPattern
|   ENDIF
|
|   IF auxiliary type is regular array THEN
|   |   determine scanning parameters
|   |   IF auxiliary is coplanar and within CFA aperture and has same element type as the CFA THEN
|   |   |   generate RxAntennaPattern
|   |   ELSE
|   |   |   generate errors for auxiliary and use to generate pattern (method TBD)
|   |   ENDIF
|   ENDIF
|
|   IF auxiliary type is arbitrary array THEN
|   |   determine scanning parameters
|   |   generate RxAntennaPattern
|   ENDIF
|
ENDLOOP by auxiliary antenna

FOR each channel
LOOP
|   IF CaptureAntennaPattern is "yes" THEN
|   |   store TxAntennaPattern or RxAntennaPattern in the intermediate product file
|   ENDIF
ENDLOOP


##### Compute Noise database #####
##### ---------------------- #####

Compute the NoiseCovar matrix between antennas
store NoiseCovar in DSPProcessingParameters file

Compute the CholeskyFactorMatrix using the NoiseCovar matrix

Compute the delta-RF matrix, which is the NoiseCovar for the current CPI minus
the NoiseCovar for the previous CPI (factor by factor difference)

##### Compute Ionospheric Model Coefficients #####
##### -------------------------------------- #####

IF Ionosphere is ON THEN
|
|   Compute geomagnetic field
|   Compute FaradayRotation
|   Compute one-way IonosphericAttenuation
|   Compute SparseIonos, the sparse sequence for ionospheric scintillation
|
ENDIF

IF the run is complete (ie this was the last CPI in the run THEN
|   close DSPProcessingParameters
ENDIF

<---return CompletionStatus to sgen

*/

}
```

```
                                                                              PAGE 7
                                      Requirements Traceability

=====================================================================================
SUBJECT
    PAGE NO.   LINE NO.      ENCLOSING FUNCTION

   NO MARKS TO REFERENCE

                                                                              PAGE 8
                                             ERROR SUMMARY

=====================================================================================
PAGE NO.   LINE NO.    ERROR TEXT

   NO ERRORS TO REPORT
```

B3.7 Burst Modelling (SBMT)

```
[][][][][][][][][][][][][][][][][][][][][][][][][][][]
[][][][]                                      [][][][]
[][][][]                                      [][][][]
[][][][]                                      [][][][]
  [][]                                          [][]
  [][]             CDADL PROCESSOR              [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]              VERSION 2.4.6               [][]
  [][]                                          [][]
  [][]         SERIAL NUMBER: C38U0011          [][]
  [][]                                          [][]
  [][]             AUTHORIZED USER:             [][]
  [][]            Raytheon - Canada             [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]                                          [][]
[][][][]                                      [][][][]
[][][][][][][][][][][][][][][][][][][][][][][][][][][]
```

```
LINE                                    sbmt.c                                  PAGE 1

/*%% Burst Modelling Task Requirements Logic */

#include "sbmt.h"

int sbmt ()   /* burst modelling task */

{

    /*+ sbmt requirements logic satisfies all functional requirements for the Burst Modelling
    Task as specified in the Software Task Definition document */

/*|

    set CompletionStatus to "successful"

    NOTE: This flag will be set to "failure" and control returned to sgen whenever an error
    is detected with any of the computations, filterings, file creations, etc.  For
    legibility, these checks are not shown at this level of pseudocode.


    ##### Compute Sparse Sequence of Orbital Positions #####
    ##### ---------------------------------------------- #####

    generate SBROrbitPosition, the sparse sequence matrix of SBR positions during the current CPI

    ##### Model Earth Clutter and add to aggregate return #####
    ##### ----------------------------------------------- #####

    initialize EarthClutterSamples

    FOR each isodop
    LOOP
    |   Compute the number of scatterers in the isodop
    |   Compute the number of range bins in the isodop
    |   FOR each range bin
    |   LOOP
    |   |   Compute the number of scatterers in the current range bin
    |   |   Initialize the antenna sparse sequence for each antenna
    |   |   FOR each scatterer
    |   |   LOOP
    |   |   |   Compute the absolute scatterer number, its range and its grazing angle
    |   |   |   Compute the R-squared propagation loss to the scatterer
    |   |   |   Compute its atmospheric attenuation
    |   |   |   Compute the scatterer location, angle and number of symmetries
    |   |   |   FOR each symmetry
    |   |   |   LOOP
    |   |   |   |   Compute clutter type, peak frequency, spectral width, scatterer reflection
    |   |   |   |   coefficient, and scatterer distribution
    |   |   |   |   Compute total precipitation attenuation, total constant loss, TxAntennaGain
    |   |   |   |   Determine burst receive start time for the current scatterer
    |   |   |   |   Compute RxAntennaGain, receive power
    |   |   |   |   IF the receive power is less than the power threshold for all antennas THEN
    |   |   |   |   ELSE
    |   |   |   |   |   generate a sparse sequence for the scatterer
    |   |   |   |   |   apply the TxAntennaGain to the sparse sequence
    |   |   |   |   |   IF ionosphere is "ON" THEN
    |   |   |   |   |   |   apply the ionospheric scintillation effects
    |   |   |   |   |   ENDIF
    |   |   |   |   |   apply precipitation attenuation and propagation loss
    |   |   |   |   |   FOR each antenna
    |   |   |   |   |   LOOP
    |   |   |   |   |   |   apply the receiving antenna gain to the receive sparse sequence
    |   |   |   |   |   |   add current sequence to the sparse antenna sequence matrix
    |   |   |   |   |   ENDLOOP by antenna
    |   |   |   |   ENDIF
    |   |   |   ENDLOOP by symmetry
    |   |   ENDLOOP by scatterer
    |   |
    |   |   FOR each antenna
    |   |   LOOP
    |   |   |   Determine the sparse sequence of burst receive sample times
    |   |   |   Determine the sparse sequence of scatterer sample times and transmit instances
|   |   |   Determine the sequence of waveform samples for the transmit instances
|   |   |   Determine the full sequence of scatterer samples
|   |   |   Determine the scatterer return
|   |   |   Add the scatterer return to EarthClutterSamples
|   |   |   FOR each delay tap
|   |   |   LOOP
|   |   |   |   Determine the sequence of waveform samples for the transmit instances
|   |   |   |   Determine the full sequence of scatterer samples
|   |   |   |   Determine the scatterer return
|   |   |   |   Add the delay tap return to EarthClutterSamples
|   |   |   ENDLOOP
|   |   ENDLOOP
|   ENDLOOP
ENDLOOP

IF CaptureClutterSignal is "Yes" THEN
|   FOR each channel
|   LOOP
|   |   store header information and EarthClutterSamples for the channel
|   |   to the intermediate products file
|   ENDLOOP
ENDIF

##### Interpolate generated samples ######
##### ----------------------------- ######

FOR each channel
LOOP
|   divide EarthClutterSamples into manageable segments
|   FOR each segment
|   LOOP
|   |   Generate the interpolated samples and store in AggregateOversamples
|   ENDLOOP
ENDLOOP

##### Model Volumetric Clutter and add to aggregate return #####
##### ---------------------------------------------------- #####

initialize Oversamples
FOR each precipitation cell
LOOP
|   FOR each cell scatterer
|   LOOP
|   |   Compute the point to point range to the scatterer
|   |   Compute total constant loss
|   |   Compute burst receive start time
|   |   Generate a sparse scatterer sequence
|   |   Apply the TxAntennaGain to the sparse sequence
|   |   IF ionosphere is "ON" THEN
|   |   |   apply ionospheric scintillation to the sparse sequence
|   |   ENDIF
|   |   Apply precipitation attenuation and propagation loss to the sparse sequence
|   |   FOR each antenna
|   |   LOOP
|   |   |   Apply RxAntennaGain to the sparse sequence
|   |   ENDLOOP
|   |   FOR each antenna
|   |   LOOP
|   |   |   Determine the sparse sequence of burst receive sample times
|   |   |   Determine the sparse sequence of scatterer sample times and transmit instances
|   |   |   Determine the sequence of waveform samples for the transmit instances
|   |   |   Determine the full sequence of scatterer samples
|   |   |   Determine the scatterer return
|   |   |   Add the scatterer return to Oversamples
|   |   |   FOR each delay tap
|   |   |   LOOP
|   |   |   |   Determine the sequence of waveform samples for the transmit instances
|   |   |   |   Determine the full sequence of scatterer samples
|   |   |   |   Determine the scatterer return
|   |   |   |   Add the delay tap return to Oversamples
|   |   |   ENDLOOP
|   |   ENDLOOP
|   ENDLOOP
ENDLOOP
```

```

FOR each channel
LOOP
|   FOR each sample
|   LOOP
|   |   add Oversamples to AggregateOversamples
|   ENDLOOP
ENDLOOP

initialize Oversamples
FOR each chaff bundle
LOOP
|   FOR each cell scatterer
|   LOOP
|   |   Compute the point to point range to the chaff bundle
|   |   Compute total constant loss
|   |   Compute burst receive start time
|   |   Generate a sparse chaff bundle sequence
|   |   Apply the TxAntennaGain to the sparse sequence
|   |   IF ionosphere is "ON" THEN
|   |   |   apply ionospheric scintillation to the sparse sequence
|   |   ENDIF
|   |   apply precipitation attenuation and propagation loss to the sparse sequence
|   |   FOR each antenna
|   |   LOOP
|   |   |   apply RxAntennaGain to sparse sequence
|   |   ENDLOOP
|   |   FOR each antenna
|   |   LOOP
|   |   |   Determine the sparse sequence of burst receive sample times
|   |   |   Determine the sparse sequence of chaff bundle sample times and transmit instances
|   |   |   Determine the sequence of waveform samples for the transmit instances
|   |   |   Determine the full sequence of chaff bundle samples
|   |   |   Determine the chaff bundle return
|   |   |   Add the chaff bundle return to Oversamples
|   |   |   FOR each delay tap
|   |   |   LOOP
|   |   |   |   Determine the sequence of waveform samples for the transmit instances
|   |   |   |   Determine the full sequence of chaff bundle samples
|   |   |   |   Determine the chaff bundle return
|   |   |   |   Add the delay tap return to Oversamples
|   |   |   ENDLOOP
|   |   ENDLOOP
|   ENDLOOP
ENDLOOP

FOR each channel
LOOP
|   FOR each sample
|   LOOP
|   |   add Oversamples to AggregateOversamples
|   ENDLOOP
ENDLOOP

##### Model Discretes and add to aggregate return #####
##### ------------------------------------------ #####

initialize Oversamples
FOR each discrete
LOOP
|   Compute discrete range, location, grazing angle, azimuth and elevation
|   Compute total precipitation attenuation
|   Compute constant loss
|   Compute burst receive start time for the discrete
|   Generate the sparse sequence for the discrete
|   Apply the TxAntennaGain to the sparse sequence
|   IF ionosphere is "ON" THEN
|   |   Apply ionospheric scintillation effects
|   ENDIF
|   Apply precipitation attenuation and propagation loss to the sparse sequence
|   FOR each antenna
|   LOOP
|   |   Apply receiving antenna gain to the sparse sequence
|   ENDLOOP
|   FOR each antenna
|   LOOP
|   |   Compute sparse sequence of burst receive sample time instances
|   |   Compute sparse sequences of discrete sample times and transmit instances
|   |   Compute sequence of waveform samples for the transmit times
|   |   Interpolate to generate a full sequence of discrete samples
|   |   Compute the discrete return
|   |   Add the discrete return to the intermediate product matrix
|   |   FOR each tap off the current antenna
|   |   LOOP
|   |   |   Compute sequence of waveform samples for the transmit times
|   |   |   Interpolate to generate a full sequence of discrete samples for the tap
|   |   |   Compute the discrete return
|   |   |   Add the discrete return to Oversamples
|   |   ENDLOOP
|   ENDLOOP
ENDLOOP

FOR each channel
LOOP
|   Add Oversamples to AggregateOversamples
ENDLOOP

##### Model Targets and add to aggregate return #####
##### ---------------------------------------- #####

FOR each target
LOOP
|   Initialize Oversamples
|   Compute target range, aspect, grazing angle, azimuth and elevation
|   Compute target reflection coefficient and distribution
|   Compute total precipitation attenuation
|   Compute constant loss
|   Compute burst receive start time for the target
|   Generate the sparse sequence for the target
|   Apply the TxAntennaGain to the space sequence
|   IF ionosphere is "ON" THEN
|   |   Apply ionospheric scintillation effects
|   ENDIF
|   Apply precipitation attenuation and propagation loss to the sparse sequence
|   FOR each antenna
|   LOOP
|   |   Apply receiving antenna gain to the sparse sequence
|   ENDLOOP
|   FOR each antenna
|   LOOP
|   |   Compute sparse sequence of burst receive sample time instances
|   |   Compute sparse sequences of target sample times and transmit instances
|   |   Compute sequence of waveform samples for the transmit times
|   |   Interpolate to generate a full sequence of target samples
|   |   Compute the target return
|   |   Add the target return to Oversamples
|   |   FOR each tap off the current antenna
|   |   LOOP
|   |   |   Compute sequence of waveform samples for the transmit times
|   |   |   Interpolate to generate a full sequence of target samples for the tap
|   |   |   Compute the target return
|   |   |   Add the target return to Oversamples
|   |   ENDLOOP
|   ENDLOOP
|   IF CaptureTargetSignal for the current target is "Yes" THEN
|   |   FOR each channel
|   |   LOOP
|   |   |   IF EnableTargetCapture for the current channel is "Yes" THEN
|   |   |   |   store Oversamples to the intermediate product file
|   |   |   ENDIF
|   |   ENDLOOP by channel
|   ENDIF
|   FOR each channel
|   LOOP
|   |   Add Oversamples to AggregateOversamples
|   ENDLOOP
ENDLOOP by target
```

```
##### Model Jammers and add to aggregate return #####
##### ---------------------------------------- #####

FOR each jammer
LOOP
|   Initialize Oversamples
|
|   ## Barrage, Swept, Follower jammers ##
|
|   IF the jammer is of type "barrage" or "swept" or "follower" THEN
|   |
|   |   IF the jammer is active during the current CPI THEN
|   |   |   Compute the start/stop time of period during which the jammer is active
|   |   ENDIF
|   |
|   |   divide the active interval down into manageable time segments
|   |
|   |   FOR each segment
|   |   LOOP
|   |   |   FOR each antenna
|   |   |   LOOP
|   |   |   |   Compute the antenna specific jammer effects modifiers
|   |   |   |   FOR each channel of the current antenna
|   |   |   |   LOOP
|   |   |   |   |   Compute the channel-specific jammer effects modifiers
|   |   |   |   |   Use the antenna- and channel-specific modifiers to generate the
|   |   |   |   |   jammer data for the current segment, and store it in Oversamples
|   |   |   |   ENDLOOP by channel
|   |   |   ENDLOOP by antenna
|   |   ENDLOOP by segment
|   |
|   |   IF the jammer is of type "swept" THEN
|   |   |   FOR each sample within the active time segment
|   |   |   LOOP
|   |   |   |   Compute SweepMod, the sweep modifier
|   |   |   |   FOR each channel
|   |   |   |   LOOP
|   |   |   |   |   Multiply the corresponding Oversamples element by SweepMod
|   |   |   |   ENDLOOP
|   |   |   ENDLOOP
|   |   ENDIF
|   |
|   |   IF the jammer is of type "follower" THEN
|   |   |   Compute the jammer's switching time
|   |   |   IF the follower jammer's frequency from the previous burst still affects the signal TH
|   |   |   |   FOR the time period from when the jammer becomes active until it switches frequenc
|   |   |   |   LOOP
|   |   |   |   |   Compute the follower modifier
|   |   |   |   |   FOR each channel
|   |   |   |   |   LOOP
|   |   |   |   |   |   Multiply the corresponding Oversamples element
|   |   |   |   |   |   by the follower modifier
|   |   |   |   |   ENDLOOP by channel
|   |   |   |   ENDLOOP by time period
|   |   |   ENDIF
|   |   ENDIF
|   |
|   |   IF the duty cycle of the jammer is less than 100 percent THEN
|   |   |   Compute those portions of the CPI during which the jammer is OFF
|   |   |   FOR each Oversamples element for which the jammer is OFF
|   |   |   LOOP
|   |   |   |   set the sample to zero
|   |   |   ENDLOOP by sample
|   |   ENDIF
|   |
|   ENDIF
|
|   ## Deception Jammers ##
|
|   IF the jammer is of type "deception" THEN
|   |
|   |   IF the RSS is to generate false targets THEN
|   |   |   FOR the defined number of false targets
|   |   |   LOOP
|   |   |   |   Compute a range offset and velocity offset for the target
|   |   |   ENDLOOP
|   |   ENDIF
|   |
|   |   FOR each false target
|   |   LOOP
|   |   |   FOR each channel
|   |   |   LOOP
|   |   |   |   Factor the false target information into Oversamples
|   |   |   ENDLOOP
|   |   ENDLOOP
|   |
|   ENDIF
|
|   IF CaptureJammerSignal for the current target is "Yes" THEN
|   |   FOR each channel
|   |   LOOP
|   |   |   IF EnableJammerCapture for the current channel is "Yes" THEN
|   |   |   |   store Oversamples to the intermediate product file
|   |   |   ENDIF
|   |   ENDLOOP by channel
|   ENDIF
|   FOR each channel
|   LOOP
|   |   Add Oversamples to AggregateOversamples
|   ENDLOOP
|
ENDLOOP by jammer

##### Model Noise and call the results the aggregate return #####
##### ----------------------------------------------------- #####

Initialize Oversamples

FOR each antenna (a)
LOOP
|   Divide the samples into manageable segments
|   FOR each segment
|   LOOP
|   |   Generate base noise values
|   |   FOR each antenna (b)
|   |   LOOP
|   |   |   IF there is a Cholesky factor effect between the two antennas THEN
|   |   |   |   FOR each tap delay of antenna (b)
|   |   |   |   LOOP
|   |   |   |   |   Factor Cholesky factor of noise covariance into noise values
|   |   |   |   ENDLOOP by tap delay
|   |   |   ENDIF
|   |   ENDLOOP by antenna (b)
|   ENDLOOP by segment
ENDLOOP by antenna

FOR each channel
LOOP
|   IF CaptureNoiseSignal is "Yes"
|   THEN
|   |   store Oversamples in the intermediate product file
|   ENDIF
ENDLOOP

FOR each channel
LOOP
|   Add Oversamples to AggregateOversamples
ENDLOOP

FOR each channel
LOOP
|   IF CaptureCompositeSignal is "yes" THEN
|   |   store AggregateOversamples in the intermediate products file
|   ENDIF
ENDLOOP

##### Apply non-linearity #####
##### ------------------- #####
```

```
FOR each channel
LOOP
|   Apply the non-linearity function to AggregateOversamples
ENDLOOP

FOR each channel
LOOP
|   IF CapturePostNonLinearitySignal is "yes" THEN
|   |   store AggregateOversamples in the intermediate products file
|   ENDIF
ENDLOOP

##### Apply IF filter and decimate #####
##### ---------------------------- #####

FOR each channel
LOOP
|   divide AggregateOversamples into manageable segments (minimum length 1024)
|   FOR each segment
|   LOOP
|   |   Factor in the IF filter values
|   |   Generate an output derivative sequence
|   |   decimate the oversampled values by selecting a subsegment
|   |   store the output segment into SRSDSamples
|   |   decimate the output derivative and store in a derived matrix
|   ENDLOOP
ENDLOOP

FOR each channel
LOOP
|   IF CaptureIFFilterOutputSignal is "yes" THEN
|   |   store SRSDSamples in the intermediate products file
|   ENDIF
ENDLOOP

##### Apply I/Q and A/D #####
##### ----------------- #####

FOR each channel
LOOP
|   Factor the I/Q distortions into SRSDSamples
ENDLOOP

FOR each channel
LOOP
|   IF CapturePostSyncDetectorSignal is "yes" THEN
|   |   store SRSDSamples in the intermediate products file
|   ENDIF
ENDLOOP

FOR each channel
LOOP
|   Factor the A/D filter values into SRSDSamples
ENDLOOP

FOR each channel
LOOP
|   IF CaptureA/DOutputSignal is "yes" THEN
|   |   store SRSDSamples in the intermediate products file
|   ENDIF
ENDLOOP

##### Process completed SRSD samples #####
##### ------------------------------ #####

Store SRSDSamples in GeneratedSRSD file

IF the run is complete (ie this was the last CPI in the run) THEN
|   FOR each of the intermediate product files
|   LOOP
|   |   close the file
|   ENDLOOP
ENDIF
```

```

<---return CompletionStatus to sgen


*/


}
```

```
                                                                    PAGE 11
                                 Requirements Traceability

============================================================================================
SUBJECT
    PAGE NO.   LINE NO.      ENCLOSING FUNCTION

   NO MARKS TO REFERENCE
```

```
                                                                        PAGE 12
                                     ERROR SUMMARY

============================================================================================
PAGE NO.   LINE NO.    ERROR TEXT

   NO ERRORS TO REPORT
```

B3.8 Playback Manager (SPMT)

```
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
[] [] [] []                                                          [] [] [] []
[] [] [] []                                                          [] [] [] []
[] [] [] []                                                          [] [] [] []
   [] []                                                                [] []
   [] []                    CDADL PROCESSOR                             [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                      VERSION 2.4.6                             [] []
   [] []                                                                [] []
   [] []              SERIAL NUMBER: C38U0011                           [] []
   [] []                                                                [] []
   [] []                    AUTHORIZED USER:                            [] []
   [] []                   Raytheon - Canada                            [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
[] [] [] []                                                          [] [] [] []
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
```

```
/*%% Signal playback manager Task Requirements Logic */

#include "spmt.h"

int spmt ()   /*% Signal Playback Manager Task */

{

    /*+ spmt requirements logic satisfies all function requirements for the Signal Playback
    Manager Task as specified in the Software Task Definition document */

/*|

    initialize interfaces to testbed controller and tdit, the RTSG executive controller task

    set RSS_state to "idle"
    set RunState to "idle"

    DO
    |
    |   wait for an incoming message
    |
    |   SWITCH on incoming message type
    |   |
    |   |   CASE StartPlaybackCommand from testbed controller
    |   |   |
    |   |   |   acknowledge the message
    |   |   |
    |   |   |   IF RSS_State is "idle" THEN
    |   |   |   |   set RSS_state to "running"
    |   |   |   |   set RunState to "running"
    |   |   |   |   initialize the error log
    |   |   |   |   send a StartingRun message to the error log
    |   |   |   |   send a LoadPlayback to tdit using playback parameters
    |   |   |   |
    |   |   |   ELSE
    |   |   |   |   send a StartPlaybackResponse message to ssgm with status
    |   |   |   |   "currently in progress"
    |   |   |   |   send a StartPlaybackRequestedWithRunInProgress message to the error log
    |   |   |   ENDIF
    |   |   |
    |   <-------break
    |   |   |
    |   |   CASE AbortPlaybackCommand from testbed controller
    |   |   |
    |   |   |   acknowledge the message
    |   |   |   IF RSS_State is "idle" THEN
    |   |   |   |   send an AbortPlaybackResponse to the testbed controller with "abort compl" status
    |   |   |   ELSE
    |   |   |   |   set RunState to "aborting"
    |   |   |   |   send an AbortPlayback to tdit
    |   |   |   ENDIF
    |   |   |
    |   |   CASE StatusMessage from testbed controller
    |   |   |
    |   |   |   acknowledge the message, giving RSS_state
    |   <-------break
    |   |   |
    |   |   CASE PingMessage from testbed controller
    |   |   |   acknowledge the message
    |   <-------break
    |   |   |
    |   |   CASE ShutdownPlaybackCommand from testbed controller
    |   |   |
    |   |   |   acknowledge the message
    |   |   |   set RunState to "terminating"
    |   |   |   IF the error log is open THEN
    |   |   |   |   send a PlaybackTerminated message to the error log
    |   |   |   |   close the error log
    |   |   |   ENDIF
    |   |   |   send an AbortPlayback message to tdit
    |   <-------break
```

```
        |   |   |
        |   |   CASE of PlaybackReady from tdit
        |   !   |   acknowledge the message
        |   |   |   send a StartPlayback message to tdit
        |   <-------break
        |   |   |
        |   |   CASE of PlaybackComplete from tdit
        |   |   |
        |   |   |   acknowledge the message
        |   |   |   send a StartPlaybackResponse to the testbed controller, providing information on
        |   |   |   timing for the current iteration
        |   |   |   send a PlaybackComplete record to the error log
        |   <-------break
        |   |   |
        |   |   CASE of PlaybackAborted from tdit
        |   |   |
        |   |   |   acknowledge the message
        |   |   |   IF RunState is "aborting" THEN
        |   |   |   |   send a PlaybackAborted message to the error log
        |   |   |   |   close the error log
        |   |   |   |   send an AbortPlaybackResponse to the testbed controller with status "abort compl"
        |   |   |   |   set RSS_State to "idle"
        |   |   |   |   set RunState to "idle"
        |   |   |   <---break
        |   |   |   ELSE
        |   |   |   !   send a PlaybackTerminated message to the error log
        |   |   |   |   close the error log
        |   |   |   |   exit
        |   |   |   ENDIF
        |   |   |
        |   ENDSWITCH
        |
        DOWHILE forever
        ENDDO


    */

    }
```

```
                                                                              PAGI
                                        Requirements Traceability
=========================================================================================
SUBJECT
    PAGE NO.    LINE NO.      ENCLOSING FUNCTION

   NO MARKS TO REFERENCE

                                                                         PAGE 4
                                              ERROR SUMMARY
=========================================================================================
PAGE NO.    LINE NO.    ERROR TEXT

   NO ERRORS TO REPORT
```

B3.9 DSP Interface (TDIT)

```
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
[] [] [] []                                                          [] [] [] []
[] [] [] []                                                          [] [] [] []
[] [] [] []                                                          [] [] [] []
   [] []                                                                [] []
   [] []                    CDADL PROCESSOR                             [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                     VERSION 2.4.6                              [] []
   [] []                                                                [] []
   [] []              SERIAL NUMBER: C38U0011                           [] []
   [] []                                                                [] []
   [] []                    AUTHORIZED USER:                            [] []
   [] []                  Raytheon - Canada                             [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
[] [] [] []                                                          [] [] [] []
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
```

tdit.c

```
/*%% Provide DSP Interface Task Requirements Logic */

#include "tdit.h"

int tdit ()   /*% Provide DSP Interface Task */

{

    /*+ tdit together with tiop requirements logic satisfies all function requirements for the Provide
    DSP Interface Task as specified in the Software Task Definition document. */

/*|

    initialize interfaces to spmt, tiop

    DO
    |
    |   Wait for an incoming message
    |
    |   SWITCH on incoming message type
    |   |
    |   |   CASE of LoadPlayback from spmt
    |   |   |
    |   |   |   acknowledge the message
    |   |   |   send a Reset command to tiop and wait for response
    |   |   |   load the identified radar file into the RTSG TPM store
    |   |   |   send a PlaybackReady to spmt
    |   <-------break
    |   |   |
    |   |   CASE of AbortPlayback from spmt
    |   |   |
    |   |   |   acknowledge the message
    |   |   |   IF playback is currently in progress THEN
    |   |   |   |   issue a Refresh to tiop to reset the data pointers to the beginning of the buffer,
    |   |   |   |   in preparation for a resumption of playback
    |   |   |   ENDIF
    |   |   |   send PlaybackAborted to spmt
    |   <-------break
    |   |   |
    |   |   CASE of StartPlayback from spmt
    |   |   |
    |   |   |   acknowledge the message
    |   |   |   send a Go command to tiop, telling it to transmit one complete iteration of the RTSG c
    |   |   |   to the ESP
    |   <-------break
    |   |   |
    |   |   CASE of IterationComplete signal from tiop
    |   |   |
    |   |   |   send PlaybackComplete message to spmt, giving timestamp and iteration number
    |   |   |   IF more iterations are needed THEN
    |   |   |   |   send a Go to tiop to transmit one complete iteration of the RTSG data to the ESP
    |   |   |   <---break
    |   |   |   ENDIF
    |   |   |
    |   ENDSWITCH
    |
    DOWHILE forever
    ENDDO

 */

 }
```

```
                                                                     PAGE 3
                                  Requirements Traceability
==================================================================================
SUBJECT
    PAGE NO.   LINE NO.      ENCLOSING FUNCTION

   NO MARKS TO REFERENCE

                                                                         PAGE 4
                                      ERROR SUMMARY
==================================================================================
PAGE NO.   LINE NO.    ERROR TEXT

   NO ERRORS TO REPORT
```

B3.10 DSP Output Task (TIOP)

```
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
[] [] [] []                                                          [] [] [] []
[] [] [] []                                                          [] [] [] []
[] [] [] []                                                          [] [] [] []
   [] []                                                                [] []
   [] []                        CDADL PROCESSOR                         [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                         VERSION 2.4.6                          [] []
   [] []                                                                [] []
   [] []                    SERIAL NUMBER: C38U0011                     [] []
   [] []                                                                [] []
   [] []                        AUTHORIZED USER:                        [] []
   [] []                       Raytheon - Canada                        [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
   [] []                                                                [] []
[] [] [] []                                                          [] [] [] []
[] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
```

```
LINE                                  tiop.c                                     PAGE 1

/*%% Provide DSP Interface Task Requirements Logic */

#include "tiop.h"

int tiop ()   /*% Provide DSP Interface Task */

{

    /*+ tiop together with tdit requirements logic satisfies all function requirements for the Provide
    DSP Interface Task as specified in the Software Task Definition document */

/*|

    initialize interfaces to tdit

    initialize data interfaces

    DO
    |
    |   wait for a Go command from tdit
    |
    |   DO
    |   |
    |   |   check for an inbound command from tdit
    |   |
    |   |   IF there is a command waiting THEN
    |   |   |   SWITCH on incoming command
    |   |   |   |
    |   |   |   |   CASE of Reset
    |   |   |   |   |   playback is finished
    |   |   |   |   |   clear data interfaces
    |   |   |   |   |   acknowledge the command
    |   |   |   <-------break
    |   |   |   |   |
    |   |   |   |   CASE of Refresh
    |   |   |   |   |   playback is finished
    |   |   |   |   |   reset internal pointers to the beginning of the run currently in progress
    |   |   |   <-------break
    |   |   |   |   |
    |   |   |   ENDSWITCH
    |   |   ENDIF
    |   |
    |   |   get new CPI parameters, including number of samples, number of channels
    |   |
    |   |   FOR each of the DSP interfaces
    |   |   LOOP
    |   |   |   FOR each portion of the CPI data
    |   |   |   LOOP
    |   |   |   |   output the data to the DSP
    |   |   |   |   wait for DSP acknowledge
    |   |   |   ENDLOOP
    |   |   ENDLOOP
    |   |
    |   DOWHILE playback not finished
    |   ENDDO
    |
    DOWHILE forever
    ENDDO

     }

 */
```

```
                                                                    PAGE 3
                                   Requirements Traceability
========================================================================================
SUBJECT
    PAGE NO.   LINE NO.      ENCLOSING FUNCTION

   NO MARKS TO REFERENCE
                                                                          PAGE 4
                                         ERROR SUMMARY
========================================================================================
PAGE NO.   LINE NO.    ERROR TEXT

   NO ERRORS TO REPORT
```

B4 TYPE DEFINITIONS AND GLOBAL DATA DICTIONARY

B4.1 Introduction

This section describes the data items referenced in the SBR RSS Software Requirements Logic, the SBR RSS CSC/CSU Communication Design and the SBR RSS Software Task Definition sections.

The RSS consists of two functional areas, Signal Generation and Signal Playback, which are functionally autonomous and independent, i.e., they exchange no data or signals within the scope of the RSS. These functional areas are treated separately throughout this document, and no attempt has been made to combine their data structures into a combined data dictionary.

The actual listing of the Global Data Definitions and Type Definitions are included as part of Appendix E, RSS PDL.

B4.1.1 RSS Data Type Definition Files

The data dictionary for Signal Generation is generated from the RSS Data Type Definition files using the C-based Design And Documentation Language (CDADL) processor. Because of the format constraints imposed by the RPCGEN utility, the Signal Playback data dictionary is presented in the form of an RPCGEN header file listing.

The RSS Data Type Definition files are:

SGEN.H	Signal Generation Typedefs
SPBK.X	Signal Playback Typedefs

B4.1.2 Interface Design Document

The typedefs and data dictionaries included in this report serves as an Interface Design Document that defines the contents and structure of information passed between CSCs/CSUs within the RSS. The source files used to produce the data dictionaries included in this report are suitable for inclusion in the actual C code implementation of the RSS thereby providing immediate reusability and ease of Software Developmental Configuration control updatability.

The source and destination of the information to be passed between the RSS CSC and external CSCs (Testbed Controller and DSP) is described in the RSS CSC/CSU Communication Design, Appendix B5.

B4.1.3 Signal Generation Data Dictionary

The Signal Generation functional area makes use of the SunOS virtual memory mapping scheme to share a common database among its CSCs/CSUs, thus all global data within this functional area is defined by a single common header file.

B4.1.3.1 CDADL Output Listing Introduction

The following sections describe the CDADL output listing included in this design note. The output listing begins with a CDADL Product banner page identifying the product name, the product version, the product serial number and the product authorized user.

B4.1.3.2 CDADL Data Dictionary Report

The Data Dictionary Report is an alphabetical listing of all data items described in the typedefs and global data source file sgen.h. Page and line numbers given in the report refer to the removed source file listings.

Data items used to complete the requirements logic appear as either concatenated, capitalized words or words joined by the underscore character.

B4.1.3.3 CDADL Error Summary Report

This report identifies any errors and warnings encountered during CDADL processing.

B5 TASK COMMUNICATIONS DESIGN

B5.1 Introduction

This section defines mechanisms and protocols used for communication by the RSS CSC. The communication paths to be considered include:

a) Interfaces between the RSS CSC and CSCs external to it
b) Interfaces between CSC/CSUs within the RSS CSC
c) Interfaces between software and hardware

B5.2 Interface between the RSS CSC and external CSCs

B5.2.1 Interface between the RSS and the Testbed Controller

The RSS and Testbed Controller will use RPCs (Remote Procedure Calls) as their primary method of inter-CSC communication. The inter-CSC messages will be passed as arguments in the remote procedure call and return an acknowledgement through the return argument in the call or by arguments passed by reference.

"RPC implements a logical client to server communications system designed specifically for the support of network applications. With RPC, the client makes a procedure call which sends requests to the server as necessary. When these requests arrive, the server calls a dispatch routine, performs whatever service is requested, sends back the reply, and the procedure call returns to the client." [1]

The logical message exchange hierarchy uses RPCs as a carrier medium. For example, during a signal generation run, the following exchange of messages will occur:

1) Testbed Controller sends a "start generation command" via RPC to the RSS. A receipt acknowledgement is returned to the Testbed Controller by the standard RPC mechanism.

2) Periodically during the generation, the RSS will issue a "start generation response" message to the Testbed Controller via the RPC mechanism, providing information on the progress of the run. A receipt acknowledgement is returned to the RSS by the standard RPC mechanism. These "start generation response" messages may be considered to be an asynchronous logical response to the earlier "start generation

---

[1] Taken from Sun Microsystems Network Programming Guide

command".

The RSS and Testbed Controller will exchange data (signal generation parameters, generated SRSD, intermediate product files, etc.) via files stored on media accessible to both CSCs. The identifiers pointing to these files will be sent by the Testbed Controller as parameters within the RPC messages. The signal generation parameter files will conform to the XDR standard and will be accessed through SunOS standard network file services.

B5.2.1.1 RSS Commands

There are two independent functional areas within the RSS CSC: Signal Generation and Signal Playback. Thus, RSS Commands breaks down within the RSS into two communications links which will be dealt with separately.

Signal Generation Commands & RSS Responses from Signal Generation

All RSS Signal Generation Commands are sent by the Testbed Controller and received by the RSS Signal Generation Manager CSC/CSU. Each command will generate an RPC acknowledgement from the RSS when it is received. Some commands also generate one or more related asynchronous messages. These messages will be covered in more detail in a later section.

Table B-3 defines the messages passed from the Testbed Controller to the RSS Signal Generation Manager CSC/CSU using RPCs.

ID	Testbed Controller Message (Signal Generation Commands)	RSS Reply	Coupling
1	Start Generation Command	RPC Acknowledge	Close
		Start Generation Response	Loose
2	Abort Generation Command	RPC Acknowledge	Close
		Abort Generation Response	Loose
3	Shutdown Generation Command	RPC Acknowledge	Close
4	Status Message	RPC Acknowledge	Close
5	Ping Message	RPC Acknowledge	Close

Table B-3 TESTBED CONTROLLER COMMANDS TO SIGNAL GENERATION MANAGER

Playback Command and RSS Responses from Playback

All RSS Playback Commands are sent by the Testbed Controller and received by the RSS Playback Manager CSC/CSU. Each command will receive an RPC acknowledgement. Some commands also generate one or more related asynchronous messages, which are described in more detail in a later section.

Table B-4 defines the messages passed from the Testbed Controller to the RSS Playback Manager CSC/CSU using RPCs.

ID	Testbed Controller Message (Playback Commands)	RSS Reply (RSS Responses from Playback)	Coupling
1	Start Playback Command	RPC Acknowledge	Close
		Start Playback Response	Loose
2	Abort Playback Command	RPC Acknowledge	Close
		Abort Playback Response	Loose
3	Shutdown Playback Command	RPC Acknowledge	Close
4	Status Message	RPC Acknowledge	Close
5	Ping Message	RPC Acknowledge	Close

Table B-4 TESTBED CONTROLLER MESSAGES TO PLAYBACK MANAGER

Signal Generation Status

Signal Generation Status messages are loosely coupled asynchronous responses to RSS Signal Generation Commands as described above. All RSS Signal Generation Status messages are sent by the RSS Signal Generation Manager CSC/CSU and received by the Testbed Controller CSC. Each command will receive an RPC acknowledgement.

Table B-5 defines the messages passed from the RSS Signal Generation Manager CSC/CSU to the Testbed Controller using RPCs.

ID	RSS Message (Signal Generation Status)	Testbed Controller Reply	Coupling
1	Start Generation Response	RPC Acknowledge	Close
2	Abort Generation Response	RPC Acknowledge	Close

Table B-5 SIGNAL GENERATION MANAGER TO TESTBED CONTROLLER MESSAGES

Playback Status

Playback Status messages are loosely coupled asynchronous responses to an earlier RSS Playback Command as described above. All RSS Playback Status messages are sent by the RSS Playback Manager CSC/CSU and received by the Testbed Controller CSC. Each command will receive an RPC acknowledgement.

Table B-6 defines the messages passed from the RSS Playback Manager CSC/CSU to the Testbed Controller using RPCs.

ID	RSS Message (Playback Status)	Testbed Controller Reply	Coupling
1	Start Playback Response	RPC Acknowledge	Close
2	Abort Playback Response	RPC Acknowledge	Close

Table B-6 PLAYBACK MANAGER TO TESTBED CONTROLLER MESSAGES

B5.2.1.2 Signal Generation Parameters

This RSS input product is stored by the Testbed Controller on disk for retrieval by the RSS. Parameters within the RSS Command "Start Signal Generation" (see above) specify to the RSS where these files are stored and their naming. The "Start Signal Generation" message provides the following information fields:

a) Sat/rad pathname. This is the full pathname (from the root) to where the satellite/radar definition file (subset of Signal Generation Parameters) is stored. If the file is stored on a remote disk (i.e., not local to the machine on which the RSS application is running), that disk is assumed to have been remote-mounted via NFS.

b) Env pathname. This is the full pathname (from the root) to where the environmental definition file (subset of Signal Generation Parameters) is stored. If the file is stored on a remote disk (i.e., not local to the machine on which the RSS application is running), that disk is assumed to have been remote-mounted via NFS.

The RSS assumes that it has exclusive rights to these files during a signal generation run, and take no action to avoid access deadlocks with the Testbed Controller CSCs.

B5.2.1.3 Signal Generation Error Log, Intermediate Results, DSP Signal Processing Parameters and Generated SRSD

These RSS output products are stored by the RSS directly onto disk. There is no direct message communications with the Testbed Controller relating to the production/output of these products. After the Testbed Controller has received a Signal Generation Response message from the RSS specifying that the signal generation run has successfully completed, the Testbed Controller assumes that these files have been created.

Intermediate Results, and DSP Signal Processing Parameters are stored in XDR standard. Generated SRSD and Signal Generation Error Log are stored as standard UNIX external files.

Parameters within the RSS Command "Start Signal Generation" (see above) specify to the RSS where these files will be stored and their naming conventions. The "Start Signal Generation" message provides the following information fields:

a) Output pathname. This is the full pathname (from the root) to where the RSS output product files (DSP Signal Processing Parameters, intermediate product files, generated SRSD data) are to be stored. If the file is stored on a remote disk (i.e., not local to the machine on which the RSS application is running), that disk is assumed to have been remote-mounted via NFS.

b) Identifier. This run identifier is included as part of all output product filenames generated by the RSS for a given run.

B5.2.1.4 Signal Playback Parameters

This RSS input product is stored by the Testbed Controller on disk for retrieval by the RSS. Parameters within the RSS Command "Start Playback" (see above) specify to the RSS where these files are stored and their naming conventions. The "Start Playback" message provides the following information fields:

a) SRSD pathname. This is the full pathname (from the root) to where the SRSD file is stored. If the file is stored on a remote disk (i.e., not local to the machine on which the RSS application is running), that disk is assumed to have been remote-mounted via NFS.

b) Iteration count. This identifies the number of times the SRSD file should be consecutively transmitted to the ESP.

B5.2.1.5 Playback Error Log

This RSS output product is stored by the RSS directly onto disk. There is no direct message communications with the Testbed Controller relating to the production/output of this product, and it is assumed by the Testbed Controller to have been created during each Playback run.

B5.2.2 Interface between RSS and ESP CSCs

The interface between the RSS and ESP CSCs is defined in the DSP/Testbed Inteface Detailed Design document [A-1], section 3.3.

The interface between the RSS and ESP CSCs will use the provided Versatile Array Signal Processor (VASP) Processing Element Support Software (PESS) Synchronous Parallel Bus (SPB) application interfaces to control the flow of information from the RSS to the ESP.

"There are five 'C' functions used to control the Synchronous Parallel Bus:

- SPB_initialize_f;
- SPB_tx_set_buf_full_f;
- SPB_tx_check_buf_empty_f;
- SPB_rx_check_buf_full_f;
- SPB_rx_set_buf_empty_f;

An application first calls SPB_initialize_f to configure the hardware for its operating mode. For each buffer to be sent in transmit mode, an application calls SPB_tx_check_buf_empty_f until buffer is available. It then loads the buffer and calls SPB_tx_set_buf_full_f to send the buffer over the SPB. For each buffer received in receive mode, an application calls SPB_rx_check_buf_full_f to poll for reception of the current double buffer over the SPB, unloads the buffer, and then calls SPB_rx_set_buf_empty_f to release the buffer for reception of further data. Double buffering is internally managed by the SPB software; the address of the next double buffer is returned from the SPB_rx_check_buf_full_f or SPB_tx_check_buf_empty_f functions." [2]

B5.3 CSC/CSU Interfaces

The RSS CSC/CSUs break into four groups:

---

[2] Taken from Design Document for the Versatile Array Signal Processor Ref: AB-DS-50-2494 Sec: 4.5.3

a) CSC/CSUs that run under SunOS (UNIX) on the Sun Workstation: Signal Generation Manager, Signal Generation Controller, Playback Manager
b) CSC/CSUs that run under the SunOS (UNIX) and make use of the CSPI coprocessor on the Sun Workstation: Initialize Run, Burst Initialization, Burst Modelling
c) CSC/CSUs that run under VxWorks on the Executive Controller board: DSP Interface
d) CSC/CSUs that run using VASP supplied software on Processing Elements (PEs) on IOPs: DSP Output

There are four different types of inter CSC/CSU communication to be considered.

a) Inter-SunOS CSC/CSU communications
b) SunOS - CSPI communications
c) VxWorks CSC/CSUs - SunOS CSC/CSU communications
d) VxWorks CSC/CSUs - VASP PE CSC/CSUs communications

B5.3.1 Inter UNIX CSC/CSU Communications Overview

There is no communications within the scope of the RSS CSC between the Signal Generation functional area (Signal Generation Manager, Generation Run Controller, Burst Initialization, Burst Modelling) and the Signal Playback functional area (Playback Manager).

B5.3.1.1 Signal Generation CSC/CSUs

Inter-process Message Queues

Communications between Signal Generation Manager and Signal Generation Controller within the Signal Generation functional area will use standard SunOS intertask message queue facilities.

The Signal Generation Manager to Signal Generation Controller message queue is described in the Table B-7.

ID	Message	Type
1	Start Generation Command	Blocked
2	Abort Generation Command	Blocked
3	Shutdown Generation Command	Unblocked

Table B-7 SIGNAL GENERATION MANAGER TO SIGNAL GENERATION CONTROLLER MESSAGES

The Signal Generation Controller to Signal Generation Manager message queue is described in Table B-8.

ID	Message	Type
1	Start Generation Response	Unblocked
2	Abort Generation Response	Unblocked

Table B-8 SIGNAL GENERATION CONTROLLER TO SIGNAL GENERATION MANAGER MESSAGES

Shared Generation Database

During a Signal Generation run, the Signal Generation Controller, Initialize Run, Burst Initialization and Burst Modelling CSC/CSUs will operate on a single virtual database, which will take advantage of SunOS's virtual memory and shared memory mapping. Individual CSC/CSUs will establish mappings to those segments of the common virtual address space they require.

For details on the SunOS shared memory mapping and virtual memory schema, refer to Sun Microsystems' System Services Overview (Part Number 800-3846-10).

B5.3.1.2 Signal Playback CSC/CSUs

Playback Manager is the only CSC/CSU within the Playback functional area which operates under SunOS.

B5.3.2 SunOS CSC/CSU to CSPI CSUs Communications Overview

The CSPI high-speed mathematical coprocessor is used by Initialize Run, Burst Initialization and Burst Modelling to achieve speed enhancements during a Signal Generation run.

A task controls the CSPI via function calls which caused execution of routines within the CSPI. The task attaches ("opens") the CSPI, specifying a data structure which will be used as a shared memory window which is used to transfer data to and from the CSPI's physical memory.

Execution of routines within the CSPI is non-blocking to the host task. The generic procedure for using the CSPI to perform a single function is as follows:

a) Open the CSPI, giving it pointers to the input and output data structures in shared memory
b) Call the appropriate function (eg: perform an FFT)
c) (Optional) Continue processing while the CSPI performs the function
d) Wait for CSPI completion
e) Close the CSPI (freeing it for use by other tasks)

B5.3.3 SunOS CSC/CSU to VxWorks CSC/CSU Communications Overview

The SunOS CSC/CSUs and VxWorks CSC/CSUs communicate via an RPC interface mechanism as described above.

Table B-9 defines the messages passed from the Playback Manager CSC/CSU to the DSP Interface Task CSC/CSU using RPCs.

B5.3.4 VxWorks CSC/CSUs - VASP IOP CSC/CSUs Communications Overview

The VASP Application Interface (Command and Response Records) software provides functions that allow a CSC/CSU operating on the Executive Controller to read and write to individual IOPs across a VME bus backplane. The functions include:

a) Get
b) Kill
c) Load
d) Open

ID	SRSD Commands	DSP Interface Task Reply	Coupling
1	Start Playback	RPC Acknowledge	Close
		Playback Complete	Loose
2	Abort Playback	RPC Acknowledge	Close
		Playback Aborted	Loose
3	Load Playback	RPC Acknowledge	Close
		Playback Ready	Loose

Table B-9 PLAYBACK MANAGER TO DSP INTERFACE MESSAGES

e) Put

See the Software Design Document for the VASP Driver Software for more details.

B6 DESIGN ERROR HANDLING LOGIC

B6.1 Introduction

The Radar Signal Simulator (RSS) is a component of the SBR SPD, which is a laboratory-based system for the demonstration and validation of radar signal processing algorithms and real-time digital signal processing hardware and as such is not a "mission critical" application. Therefore, it will not provide any fault tolerant capability. The RSS will assume after its initial startup that all RSS hardware is functioning as expected. The RSS will report all unrecoverable system errors if possible to the Testbed Controller through the Error Log (severe arithmetic exceptions, unrecoverable COTS errors, bus errors). The use of defensive coding practices will avoid most or all non-severe errors from occurring. Should such an error occur, the RSS will report it through the error log.

B6.1.1 Error Handling Overview

Five sources of errors will be encountered by the RSS:

a) SunOS Errors
b) CSPI Errors
c) VxWorks Errors
d) VASP Errors
e) RSS Application Errors

B6.1.2 Logging

In order to maintain commonality of messages logged to error log files within the SBR, the RSS will adhere to the following standard error line format:

<time-stamp> <function-name> <severity-code> <message-text>

"time-stamp": indicates the time at which the error message was output (ie: 12:10:15).

"function-name": identifies the name of the function logging the error message (ie:f_crad_validate_alt")

"severity-code": indicates the severity of the error message (one of "FATAL", "WARN", and "INFO")

"message-text": provides the complete error message text clearly identifying the condition at fault (ie: "invalid altitude: -3000")

Normally this will fit on an 80 character line. If the <message-text> field is too long, it will be wrapped onto the next line.

B6.2 Sun OS Errors

SunOS system service provide standardized error signal returns when an error condition is encountered. The application may specify a handler routine to catch and process the signal, block or ignore the signal or specify default signal handling. Signals are defined in the standard system include file <signal.h>. A more detailed description of signals, signal handlers, timers, etc. may be found in chapter 3 of the SunOS System Services Overview manual.

Signals fall into five classes:

a) hardware generated signals such as floating point or arithmetic exceptions and memory violations
b) resource control such as limitation of file size creations
c) software generated signals such as expiration of interval timers or user-generated signals such as "quit"
d) input/output notification such as notice that an asynchronous I/O operation has completed
e) process control such as process activation/halt signals

Only the first two signal types are considered errors for the purposes of this document. The standard RSS error handling philosophy will be for the application attempt to intercept all error signals which will by default result in program termination, and handle them via application-level signal handlers.

B6.2.1 Hardware Signals

"Unexpected" hardware signals, such as floating point exceptions, will be anticipated by the applications code as far as possible and avoided in a context-appropriate manner. In many cases, a floating point exception will result in an individual sample being set to zero (i.e., essentially ignored). Should an unexpected signal occur outside of the scope of these checks, the application code will trap it, log the occurrance to the appropriate error log, then terminate gracefully.

B6.2.2 Resource Control Signals

The most common resource error which the RSS application may expect to encounter is running out of disk storage. The RSS signal generation software will try to predict its resource needs at the beginning of a signal generation run and confirm that sufficient resources are available. If the RSS application encounters a lack of disk storage at any point during the run, it will log this problem to the error log and terminate the run.

B6.3 CSPI Errors

The CSPI returns status codes to the calling task. A CSPI failure or CSPI unavailability will be considered a fatal error by the RSS, and will cause the currently in-progress Signal Generation run to log the problem and terminate.

B6.4 VxWorks Errors

The general mechanism of VxWorks error handling is described in detail in the Appendix C of the DSP Detailed Design Document [R-?]. Those portions of the RSS Playback application software which execute within the VxWorks environment will follow a comparable error handling scheme.

B6.5 VASP Errors

The general mechanism of VASP error handling is described in detail in the Appendix C of the DSP Detailed Design Document [R-?]. Those portions of the RSS Playback application software which execute within the VASP environment will follow a comparable error handling scheme.

B6.6 RSS Application Errors

The details of the handling of individual RSS Application Errors is TBD during detailed design.

B7 CSU DESIGN

B7.1 Signal Generation Manager CSU (SSGM)

B7.1.1 Introduction

The Signal Generation Manager (ssgm) CSC/CSU is the initial signal generation task, which is started by the Testbed Controller. It starts the remaining tasks in the signal generation package then serves as a communications interface between the RSS and the testbed controller.

It listens for RPC messages from the Testbed Controller and forwards them as IPC messages to sgen, the main generation task. It also takes incoming IPC messages from sgen, and forwards them as RPCs to the testbed controller.

B7.1.2 CSC/CSU Logic Flow Diagrams

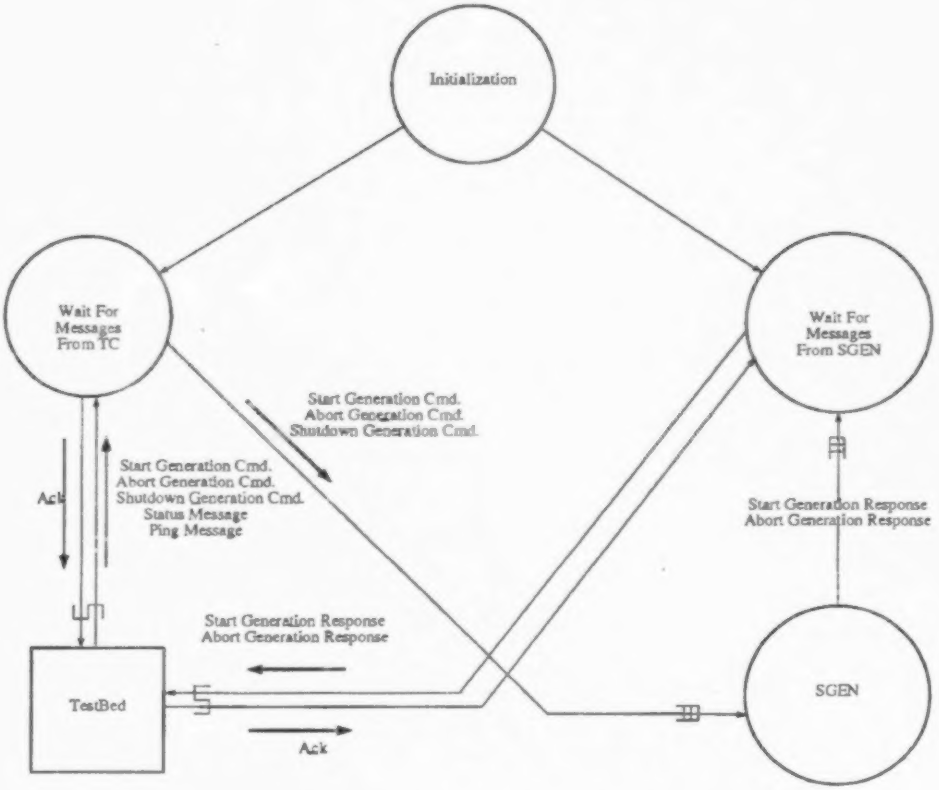

Initialization
Wait For Messages From TC
Wait For Messages From SGEN
Start Generation Cmd.
Abort Generation Cmd.
Shutdown Generation Cmd.
Ack
Start Generation Cmd.
Abort Generation Cmd.
Shutdown Generation Cmd.
Status Message
Ping Message
Start Generation Response
Abort Generation Response
Start Generation Response
Abort Generation Response
TestBed
Ack
SGEN

Figure B-4 SIGNAL GENERATION MANAGER (SSGM) LOGIC FLOW

B7.1.3 Module Design

B7.1.3.1 Design Logic

The design logic for each module shown in the Signal Generation Manager CSU Module Invocation Tree is contained in Appendix E together with applicable design reports produced by CDADL.

`Module Invocation Hierarchy`

`See Appendix E for invocation hierarchy.`

B7.1.4 Module Test Plan

Module Name	Desk Inspection	Functional Test	Performance Analysis
f_ssgm_main	X	X	
f_ssgm_initialize	X	X	
i_ssgm_cleanup	X	X	
f_ssgm_tc_inbound	X	X	

Table B-10 SSGM MODULE TEST PLAN

Table B-10 shows the module test plan for SSGM.

B7.1.5 Signal Generation Manager CSU Test Plan

Objective: Verify that the correct actions are performed for each input message.

Case 1:
Verify that Initialization is correctly performed.

Case 2:
Verify that the task is continually ready to receive incoming RPCs from the Testbed Controller.
Verify that the task continually checks its IPC receive queue for messages from sgen.

Case 3:

Verify that the correct actions are taken in response to each Command Request from the Testbed Controller:

Start Generation Command and Abort Generation Command:

- Check that these messages are placed in the outbound IPC message queue.
- Check that ssgm sends the RPC acknowledge back to the Testbed Controller.

Shutdown Generation Command:

- Check that this message is placed in the outbound IPC message queue.
- Check that ssgm sends an RPC acknowledge back to the Testbed Controller.
- Check that ssgm is terminated.

Status Message:

- Check that the current RSS state is sent back to Testbed Controller in the RPC acknowledge.

Ping Message:

- Check that an RPC acknowledge is returned to the Testbed Controller.

Case 4:

Verify correct processing of Start Generation Response and Abort Generation Response messages from sgen:

- Check that the test message is read and cleared from the inbound IPC message queue.
- Check that ssgm packages this message correctly as an RPC and forwards it to the Testbed Controller.

B7.2 Signal Generation Controller CSU ($GEN)

B7.2.1 Introduction

The Signal Generation Controller (sgen) CSC/CSU is spawned by ssgm at startup to manage the Signal Generation activity. It, in turn, spawns three child processes (srun, sbit, sbmt) to perform the actual generation activity.

The Signal Generation Controller initializes itself into an "idle" state. While in an "idle" state, it continually monitors an incoming message queue for messages from ssgm, waiting for a Start Generation command forwarded from the testbed controller, or a Shutdown Generation command, telling it to self-terminate.

When sgen receives a Start Generation command, it enters a "running" state, and spawns its initial child process (srun) to begin the Generation activity.

While it is in a "running" state, sgen continually monitors the message queue for incoming commands (Abort Generation or Shutdown Generation) from the Testbed Controller, and an internal state indicator which is shared by it and its child processes.

a) When sgen receives an Abort Generation command while "running", it signals its currently-running child process to terminate, then returns to an "idle" state.

b) When sgen receives a Shutdown Generation command while "running", it signals its running child process to terminate, then terminates itself.

c) Whenever it detects a state change, indicating failure or completion of a child process, it responds accordingly with a response message to ssgm (and through it, to the Testbed Controller) and/or by spawning a new child process to continue the Generation activity.

B7.2.2 CSC/CSU Logic Flow Diagrams

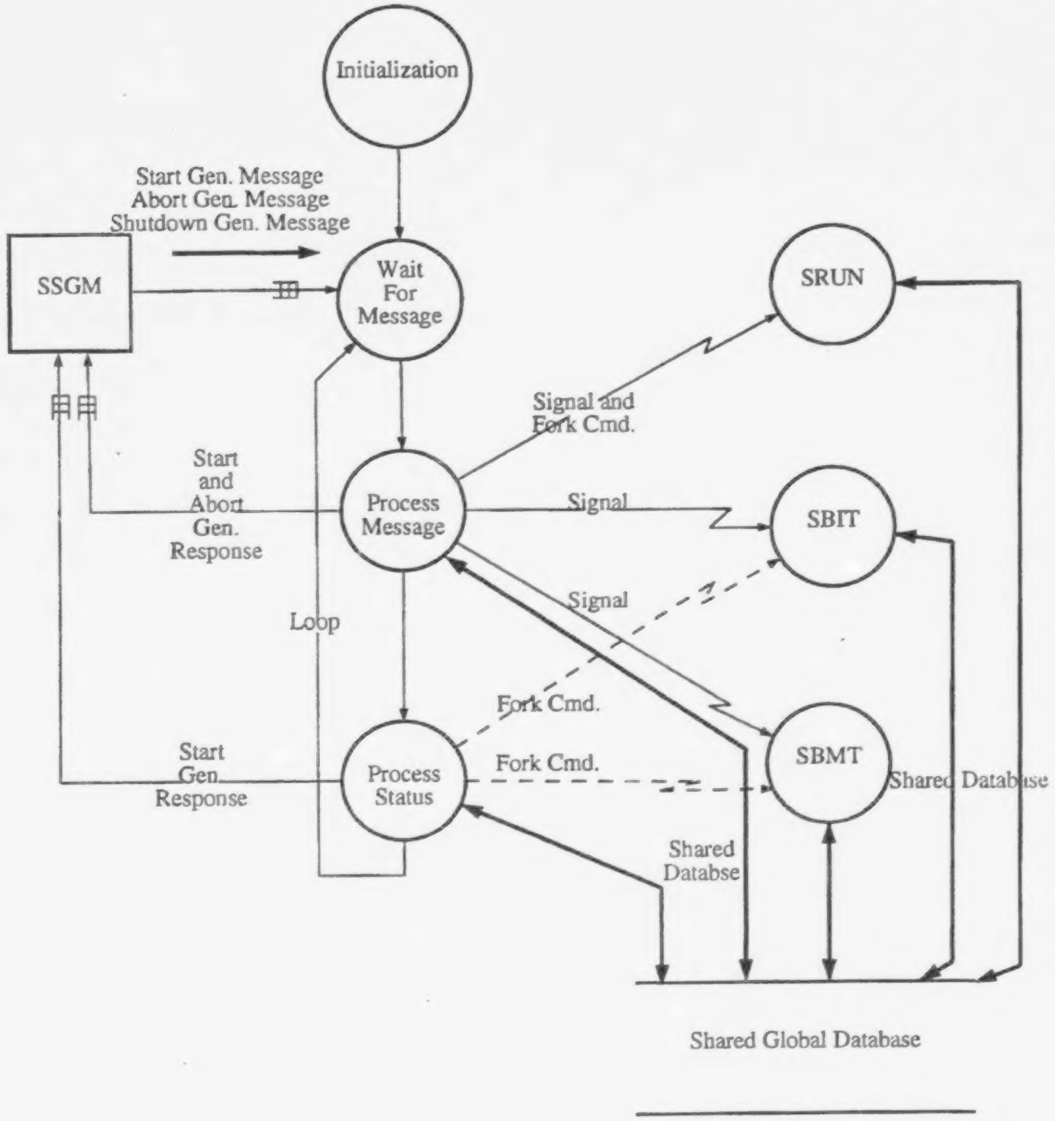

Initialization
Start Gen. Message
Abort Gen. Message
Shutdown Gen. Message
SSGM
Wait For Message
SRUN
Signal and Fork Cmd.
Start and Abort Gen. Response
Process Message
Signal
SBIT
Signal
Loop
Fork Cmd.
Start Gen. Response
Process Status
Fork Cmd.
SBMT
Shared Database
Shared Databse
Shared Global Database

Figure B-5 SIGNAL GENERATION CONTROLLER (SGEN) LOGIC FLOW

B7.2.3 Module Design

B7.2.3.1 Design Logic

The design logic for each module shown in the Signal Generation Controller CSU Module Invocation Tree is contained in Appendix E together with applicable design reports produced by CDADL.

Detailed design information on srun, sbit, and sbmt are included in separate documents.

`Module Invocation Hierarchy`

`See Appendix E for invocation hierarchy.`

B7.2.4 Module Test Plan

Table B-11 shows the module test plan for SSGM.

Module Name	Desk Inspection	Functional Test	Performance Analysis
f_sgen_main	X	X	
f_sgen_initialize	X	X	
f_sgen_terminate	X	X	
f_sgen_failed	X	X	
f_sgen_abort	X	X	

Table B-11 SGEN MODULE TEST PLAN

B7.2.5 Signal Generation Controller CSU Test Plan

Objective: Verify that the correct actions are performed for each input message or change of state.

Case 1:

Verify that Initialization is correctly performed and the task comes up in an "idle" state.

Case 2:

Verify that the task continually checks its IPC receive queue for messages from ssgm and clears these messages after they have been read.

Case 3:

Verify that the correct actions are taken in response to each Command Request from the Testbed Controller (via ssgm):

Start Generation Command
- Verify that sgen does the following:
  - if "running", reports an error to the TC and the log, and continues
  - if "idle", switches to "running", issues a Start Generation Response message to ssgm, and correctly spawns srun to begin the generation

Abort Generation Command:
- Verify that sgen does the following:
  - if "idle", issues an Abort Generation Response message to ssgm
  - if "running", brings down the currently-active child process, issues an Abort Generation Response message to ssgm, and switches to "idle"

Shutdown Generation Command:
- Verify that sgen does the following:
  - if "idle", terminates
  - if "running", brings down the currently-active child process, then terminates

Case 4:

Verify correct processing of status changes from the child processes.

Completion status from srun:
- Verify that sgen spawns sbit

Completion status from sbit:
- Verify that sgen spawn sbmt

Completion status from sbmt:
- Verify that sgen generates a Start Generation Response message to ssgm, reporting a valid "percentage complete"
- If the run is not complete, verify that sgen spawns sbit
- If the run is complete, verify that sgen returns to "idle" state

B7.3 Run Initialization CSU (SRUN)

B7.3.1 Introduction

The Run Initialization (srun) CSC/CSU is spawned by sgen at the start of a Signal Generation activity. It performs all those functions within the Signal Generation activity which are done once at the beginning of a run.

Run Initialization uses the information provided by the Signal Generation Parameters files to build a run-constant database which will be used by Burst Initialization (sbit) and Burst Modelling (sbmt) during subsequent processing.

B7.3.2 CSC/CSU Logic Flow Diagrams

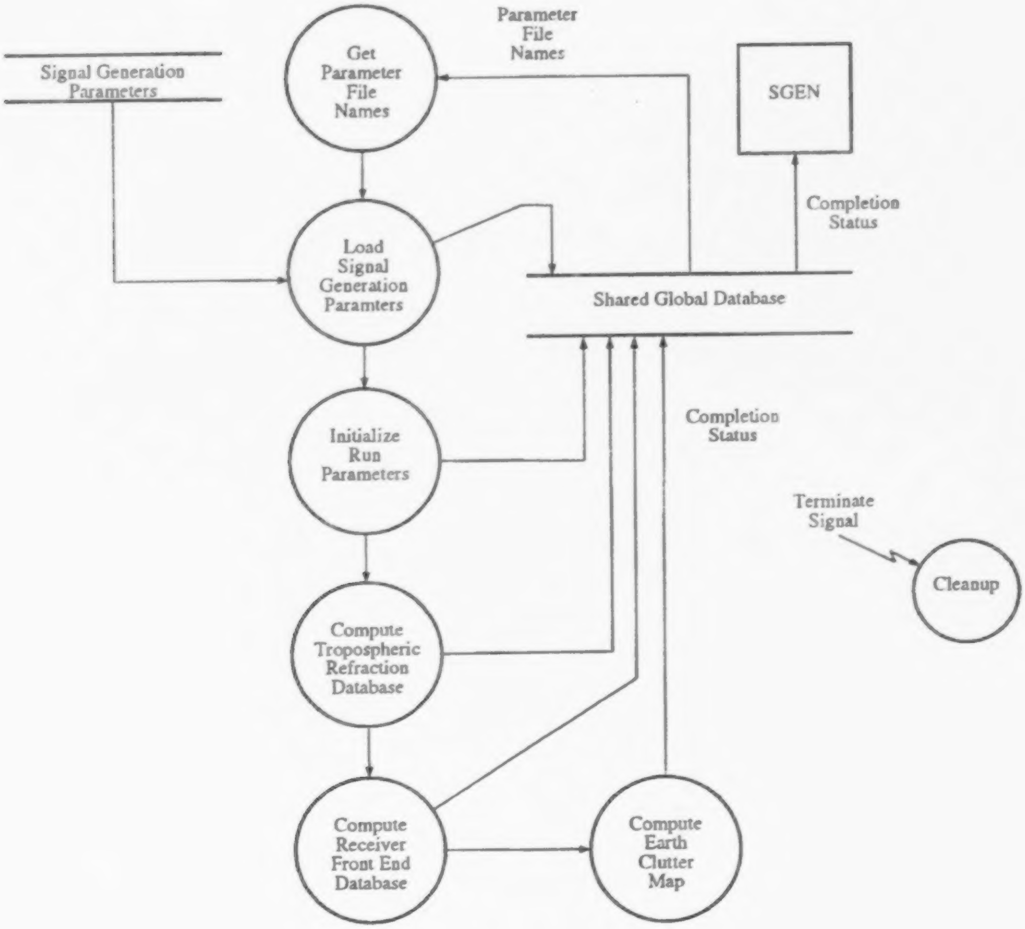

Parameter File Names
Signal Generation Parameters
Get Parameter File Names
SGEN
Completion Status
Load Signal Generation Paramters
Shared Global Database
Initialize Run Parameters
Completion Status
Terminate Signal
Cleanup
Compute Tropospheric Refraction Database
Compute Receiver Front End Database
Compute Earth Clutter Map

Figure B-6 RUN INITIALIZATION (SRUN) LOGIC FLOW

B7.3.3 Module Design

B7.3.3.1 Design Logic

The design logic for each module shown in the Run Initialization CSU Module Invocation Tree is contained in Appendix E with the applicable design reports produced by CDADL.

```
Module Invocation Hierarchy

See Appendix E for invocation hierarchy.
```

B7.3.4 Module Test Plan

Table B-12 shows the module test plan for SRUN.

Module Name	Desk Inspection	Functional Test	Performance Analysis
f_srun_main	X	X	
f_srun_initialize	X	X	
f_srun_init_ssp	X	X	
f_srun_seeds	X	X	
f_srun_gen_seeds	X	X	
i_srun_cleanup	X	X	
f_srun_creat_srsd	X	X	
f_srun_load_trdb	X	X	
f_srun_rx_db	X	X	
f_srun_iff	X	X	
f_srun_filter_store	X	X	
f_srun_ec_map	X	X	X
f_srun_det_patch_valid	X	X	
f_srun_mercator	X	X	
f_srun_straddle_check	X	X	
f_srun_orbit_coeffs	X	X	
f_srun_getparams	X	X	
f_srun_straddle_points	X	X	

Table B-12 SRUN MODULE TEST PLAN

B7.3.5 Run Initialization Test Plan

Objective: Verify that srun performs all activities which are required on a one-shot basis at the beginning of a Signal Generation run.

Case 1:
Verify that the task begins processing upon being spawned by sgen.

Case 2:
Verify that the task loads the Signal Generation Parameters from external UNIX files.

Case 3:
Verify that the task terminates upon completion, leaving behind the run-constant database in shared (virtual) memory.

Case 4:
Verify that the task responds to an abort signal from sgen by performing necessary cleanup activities, logging the premature termination to the error log, reporting the "terminated" status to sgen, then terminating.

Case 5:
Verify that the task makes correct use of the error log.

B7.4 Burst Initialization CSU (SBIT)

B7.4.1 Introduction

The Burst Initialization (sbit) CSC/CSU is spawned by sgen. It performs all those functions within the Signal Generation activity which are done once at the beginning of each CPI during the course of the Generation.

Burst Initialization uses the information generated by srun, and builds a burst database which will be used by Burst Modelling (sbmt) during subsequent processing of the CPI.

B7.4.2 CSC/CSU Logic Flow Diagrams

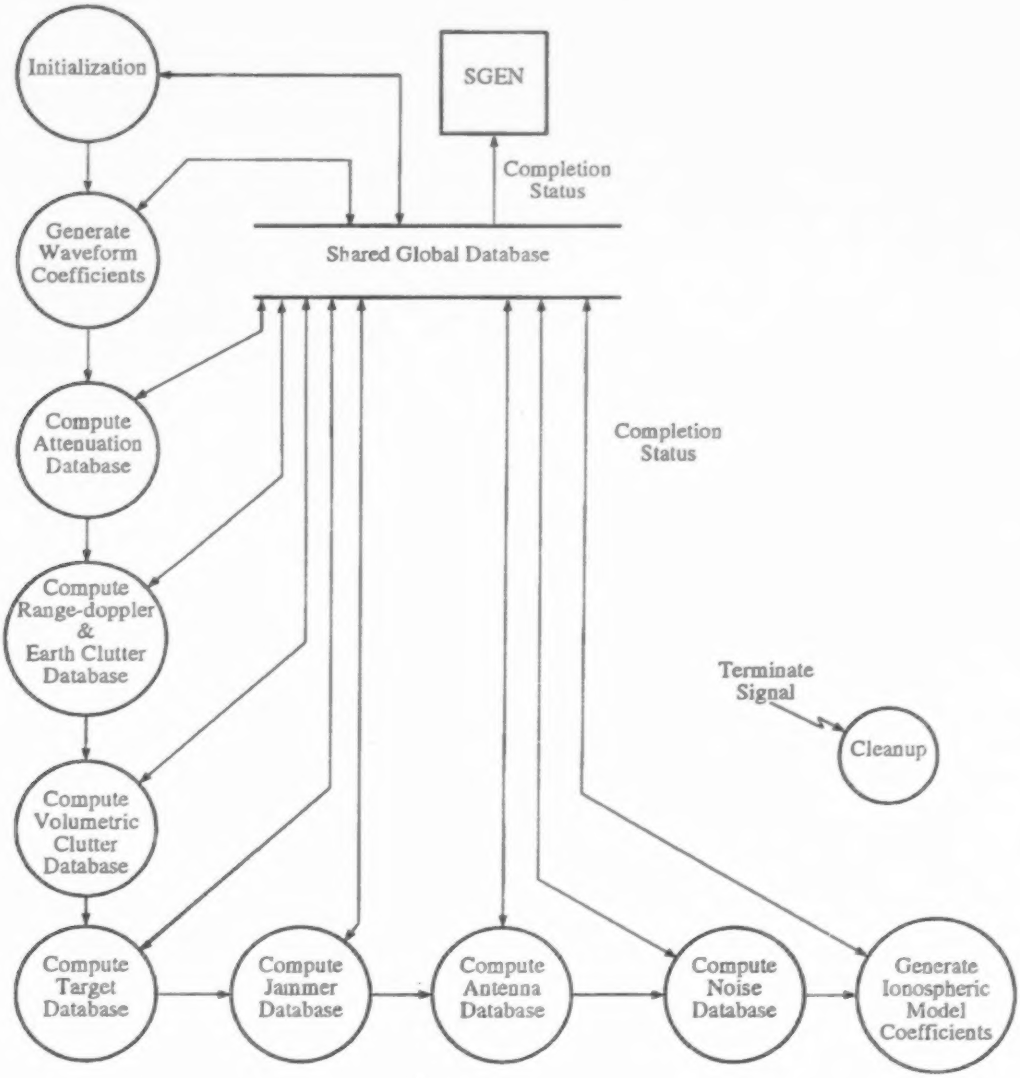

Initialization
SGEN
Completion Status
Generate Waveform Coefficients
Shared Global Database
Compute Attenuation Database
Completion Status
Compute Range-doppler & Earth Clutter Database
Terminate Signal
Cleanup
Compute Volumetric Clutter Database
Compute Target Database
Compute Jammer Database
Compute Antenna Database
Compute Noise Database
Generate Ionospheric Model Coefficients

Figure B-7 BURST INITIALIZATION (SBIT) LOGIC FLOW

B7.4.3 Module Design

B7.4.3.1 Design Logic

The design logic for each module shown in the Burst Initialization CSU Module Invocation Tree is contained in Appendix E together with applicable design reports produced by CDADL.

Module Invocation Hierarchy

See Appendix E for invocation hierarchy.

B7.4.4 Module Test Plan

Table B-13 shows the module test plan for SBIT.

Module Name	Desk Inspection	Functional Test	Performance Analysis
f_sbit_main	X	X	
f_sbit_initialize	X	X	
i_sbit_cleanup	X	X	
f_sbit_atten	X	X	X
f_sbit_antenna	X	X	X
f_sbit_beam_centre	X	X	
f_sbit_cell_valid	X	X	
f_sbit_cgep_store	X	X	
f_sbit_ionosphere	X	X	
f_sbit_jammer	X	X	
f_sbit_target	X	X	
f_sbit_noise	X	X	
f_sbit_rd_map	X	X	X
f_sbit_precip_db	X	X	X
f_sbit_precip_valid	X	X	
f_sbit_chaff_db	X	X	
f_sbit_waveform	X	X	

Table B-13 SBIT MODULE TEST PLAN

B7.4.5 Burst Initialization Test Plan

Objective: Verify that sbit performs all activities which are required on a one-shot basis at the beginning of a Signal Generation run.

Case 1:
Verify that the task begins processing upon being spawned by sgen.

Case 2:
Verify that the task gains access to the database generated by srun.

Case 3:
Verify that the task terminates upon completion, leaving behind the burst database in shared (virtual) memory.

Case 4:
Verify that the task responds to an abort signal from sgen by performing necessary cleanup activities, logging the premature termination to the error log, reporting the "terminated" status to sgen, then terminating.

Case 5:
Verify that the task makes correct use of the error log.

B7.5 Burst Modelling CSU (SBMT)

B7.5.1 Introduction

The Burst Modelling (sbmt) CSC/CSU is spawned by sgen. It performs all those functions within the Signal Generation activity which are done once at the beginning of each CPI during the course of the Generation.

Burst Modelling uses the information generated by srun to build a burst database which will be used by Burst Modelling (sbmt) during subsequent processing of the CPI.

B7.5.2 CSC/CSU Logic Flow Diagrams

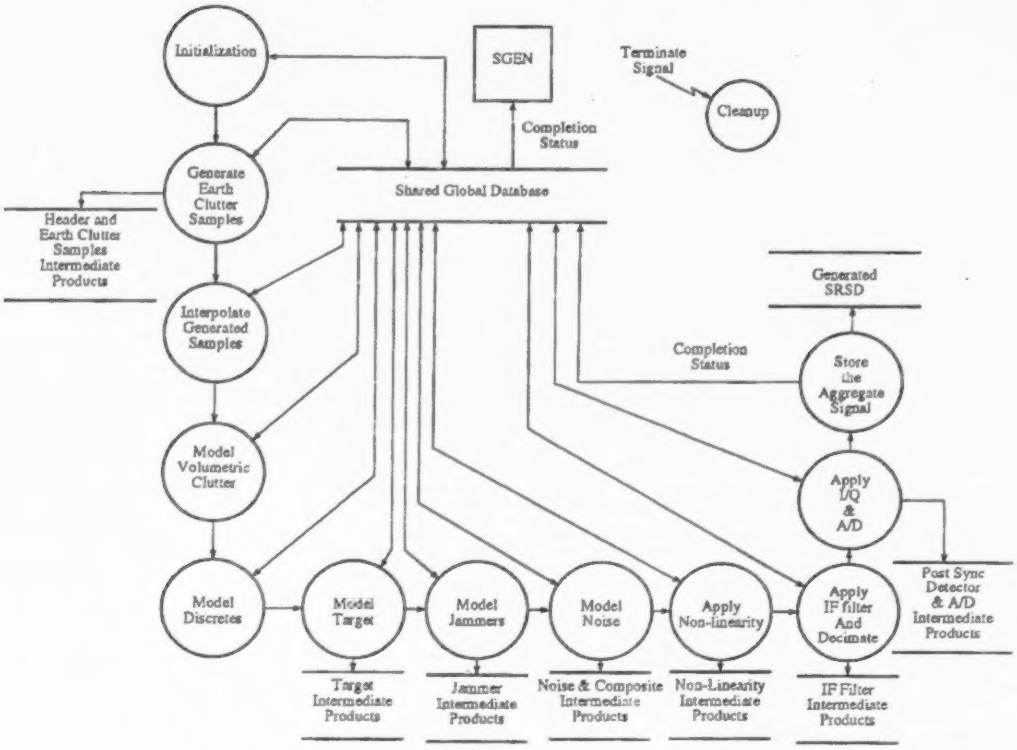

Initialization
SGEN
Terminate Signal
Cleanup
Completion Status
Generate Earth Clutter Samples
Shared Global Database
Header and Earth Clutter Samples Intermediate Products
Generated SRSD
Interpolate Generated Samples
Completion Status
Store the Aggregate Signal
Model Volumetric Clutter
Apply I/Q & A/D
Post Sync Detector & A/D Intermediate Products
Model Discretes
Model Target
Model Jammers
Model Noise
Apply Non-linearity
Apply IF filter And Decimate
Target Intermediate Products
Jammer Intermediate Products
Noise & Composite Intermediate Products
Non-Linearity Intermediate Products
IF Filter Intermediate Products

Figure B-8 BURST MODELLING (SBMT) LOGIC FLOW

B7.5.3 Module Design

B7.5.3.1 Design Logic

The design logic for each module shown in the Burst Modelling CSU Module Invocation Tree is contained in Appendix E together with applicable design reports produced by CDADL.

`Module Invocation Hierarchy`

`See Appendix E for invocation hierarchy.`

B7.5.4 Module Test Plan

Table B-14 shows the module test plan for SBMT.

Module Name	Desk Inspection	Functional Test	Performance Analysis
f_sbmt_main	X	X	
i_sbmt_cleanup	X	X	
f_sbmt_ad	X	X	
f_sbmt_discretes	X	X	X
f_sbmt_eclutter	X	X	X
f_sbmt_if_filter	X	X	
f_sbmt_interpolate	X	X	
f_sbmt_iq	X	X	
f_sbmt_jammers	X	X	X
f_sbmt_noise	X	X	X
f_sbmt_non_linear	X	X	
f_sbmt_sparse_orbit	X	X	
f_sbmt_store	X	X	
f_sbmt_store_int	X	X	
f_sbmt_targets	X	X	X
f_sbmt_precip	X	X	
f_sbmt_chaff	X	X	

Table B-14 SBMT MODULE TEST PLAN

B7.6 Burst Modelling Test Plan

Objective: Verify that sbmt performs all activities which are required to generate one CPI worth of signal data and selected intermediate products.

Case 1:
Verify that the task begins processing upon being spawned by sgen.

Case 2:

Verify that the task gains access to the database generated by srun and sbit.

Case 3:

Verify that the task terminates upon completion, leaving behind the SRSD and selected intermediate product files.

Case 4:

Verify that the task responds to an abort signal from sgen by performing necessary cleanup activities, logging the premature termination to the error log, reporting the "terminated" status to sgen, then terminating.

Case 5:

Verify that the task makes correct use of the error log.

B7.7 RSS Utilities CSU (SUTL)

B7.7.1 Introduction

The RSS Utilities (sutl) CSC/CSU is a collection of utility functions called by other RSS tasks to perform common functions. Because the RSS breaks down into two unrelated subsystems (Signal Generation and Signal Playback), the only common utilities are related to error logging functions.

B7.7.2 Module Design

B7.7.2.1 Design Logic

The design logic for each module shown in the RSS Utilities CSU Module Invocation Tree is contained in Appendix E together with applicable design reports produced by CDADL.

`Module Invocation Hierarchy`

`See Appendix E for invocation hierarchy.`

B7.7.3 Module Test Plan

Table B-15 shows the module test plan for SUTL.

Module Name	Desk Inspection	Functional Test	Performance Analysis
f_sutl_olog	X	X	
f_sutl_elog	X	X	
f_sutl_clog	X	X	

Table B-15 SUTL MODULE TEST PLAN

B7.7.4 RSS Utilities CSU Test Plan

Objective: Verify that the correct actions are performed for each invocation.

Case 1:

Verify that a call to f_sutl_olog() creates an external UNIX file and returns the file handle.

Case 2:

- Verify that a call to f_sutl_elog() adds the required record to the external file indicated by the file handle.
- Verify that a similar call with parameter substitutions generates a valid log record.

Case 3:

Verify that a call to f_sutl_clog() closes the log file.

B7.8 Signal Playback Manager CSU (SPMT)

B7.8.1 Introduction

The Signal Playback Manager (spmt) CSC/CSU is started by the Testbed Controller. It establishes communications with the DSPI RPC Server (trsv) then serves as a communications interface between the RTSG and the testbed controller.

It listens for RPC messages from the Testbed Controller and translates them into commands to trsv. It also takes incoming RPC messages from trsv, and forwards them as RPCs to the testbed controller.

B7.8.2 CSC/CSU Logic Flow Diagrams

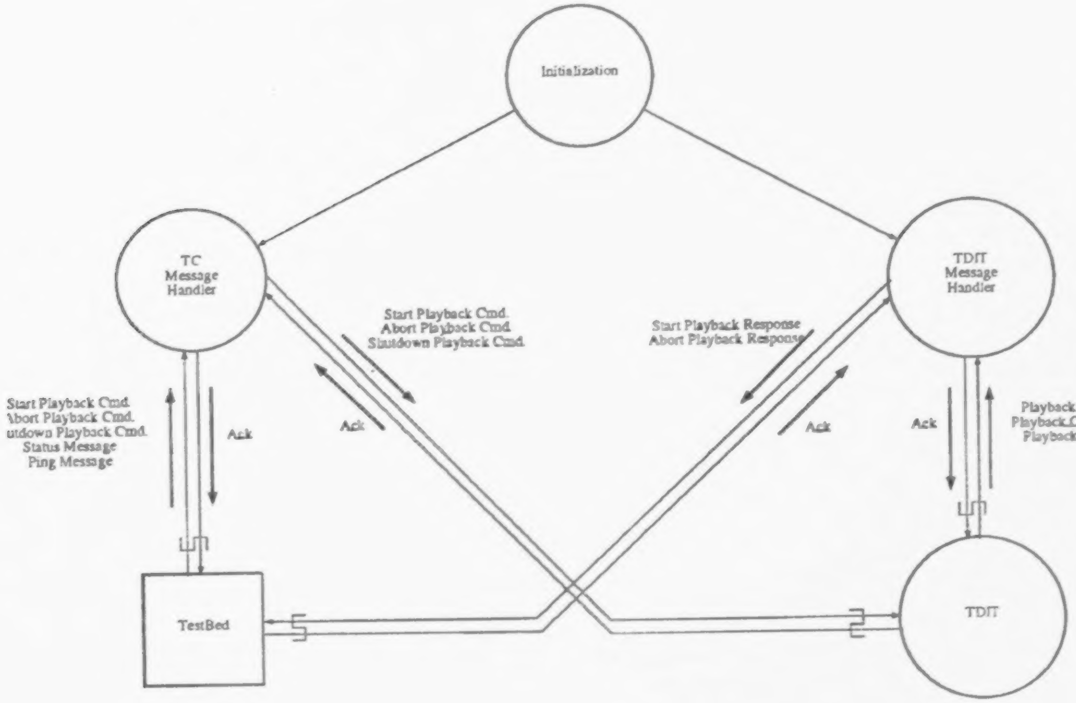

Initialization
TC Message Handler
TDIT Message Handler
Start Playback Cmd.
Abort Playback Cmd.
Shutdown Playback Cmd.
Start Playback Response
Abort Playback Response
Start Playback Cmd.
Abort Playback Cmd.
utdown Playback Cmd.
Status Message
Ping Message
Ack
Ack
Ack
Ack
Playback Ready
Playback Complete
Playback Abort
TestBed
TDIT

Figure B-9 SIGNAL PLAYBACK MANAGER (SPMT) LOGIC FLOW DIAGRAM

B7.8.3 Module Design

B7.8.3.1 Design Logic

The design logic for each internal module shown in the Signal Playback Manager CSU Module Invocation Tree is contained in Appendix 1 together with applicable design reports produced by CDADL. The Signal Playback Manager CSU also uses modules from the RSS Utilities package. Refer to the detailed design for this package for CDADL listings of these modules.

```
Module Invocation Hierarchy

CDADL PAGE#  LINE#  Module              (See Appendix E)

See Appendix E for invocation hierarchy.
```

B7.8.3.2 Module Test Plan

Table B-16 shows the module test plan for SPMT.

Module Name	Desk Inspection	Functional Test	Performance Analysis
f_spmt_main	X	X	
f_spmt_initialize	X	X	
f_spmt_tc_hndlr	X	X	
f_spmt_tdit_hndlr	X	X	

Table B-16 SPMT MODULE TEST PLAN

B7.8.4 Signal Playback Manager CSU Test Plan

Objective: Verify that the correct actions are performed for each input message.

Case 1:
Verify that Initialization is correctly performed.

Case 2:
Verify that the task is continually ready to receive incoming RPCs from the Testbed

Controller and the DSPI RPC Server task.

Case 3:

Verify that the correct actions are taken in response to each Command Request from the Testbed Controller:

Start Playback Command:

- Check that the Load Playback RPC message is forwarded to the DSPI RPC Server.
- Check that spmt sends the RPC acknowledge back to the Testbed Controller.

Abort Playback Command:

- Check that the Abort Playback RPC message is forwarded to the DSPI RPC Server.
- Check that spmt sends the RPC acknowledge back to the Testbed Controller.

Shutdown Playback Command:

- Check that the Abort Playback RPC message is forwarded to the DSPI RPC Server.
- Check that spmt sends an RPC acknowledge back to the Testbed Controller.
- Check that spmt is terminated.

Status Message:

- Check that the current RSS Playback state is sent back to Testbed Controller in the RPC acknowledge.

Ping Message:

- Check that an RPC acknowledge is returned to the Testbed Controller.

Case 4:

Verify that the correct actions are taken in response to each incoming RPC from the RTSG:

Playback Ready:

- Check that an RPC acknowledge is returned to the DSPI RPC Server.
- Check that the Start Playback RPC message is forward to the DSPI RPC Server.

Playback Complete:

- Check that an RPC acknowledge is returned to the DSPI RPC Server.
- Check that the Start Playback Response RPC message is forward to the Testbed Controller.

Playback Aborted:

- Check that an RPC acknowledge is returned to the DSPI RPC Server.
- Check that the Playback Aborted RPC message is forward to the Testbed Controller.

Case 5:

Examine the error log maintained by spmt, and verify that it has logged all activities throughout the test sequence.

B7.9 DSP Interface CSU (TDIT)

B7.9.1 Introduction

The DSP Interface CSC/CSU controls the overall operation of the RTSG. It receives and executes SRSD Commands sent by DSPI RPC Server and acknowledge these commands. DSP Interface initiates the loading of run parameters and SRSD Data to the SRSD Sender CSC/CSU in response to a LoadPlayback command. It then waits for a StartPlayback command to initiate the sending of SRSD Data to the DSP. As well, it can in response to an AbortPlayback Command abort the sending of SRSD Data. It also provides a mechanism whereby it sends a Iteration Complete message along with timing information to the Playback Manager.

B7.9.2 CSC/CSU Data Flow Diagrams

The ESP CSC/CSU Context Diagram is extracted from the ESP Inter-CSC/CSU Communication Diagram found in the ESP CSC Definition design note.

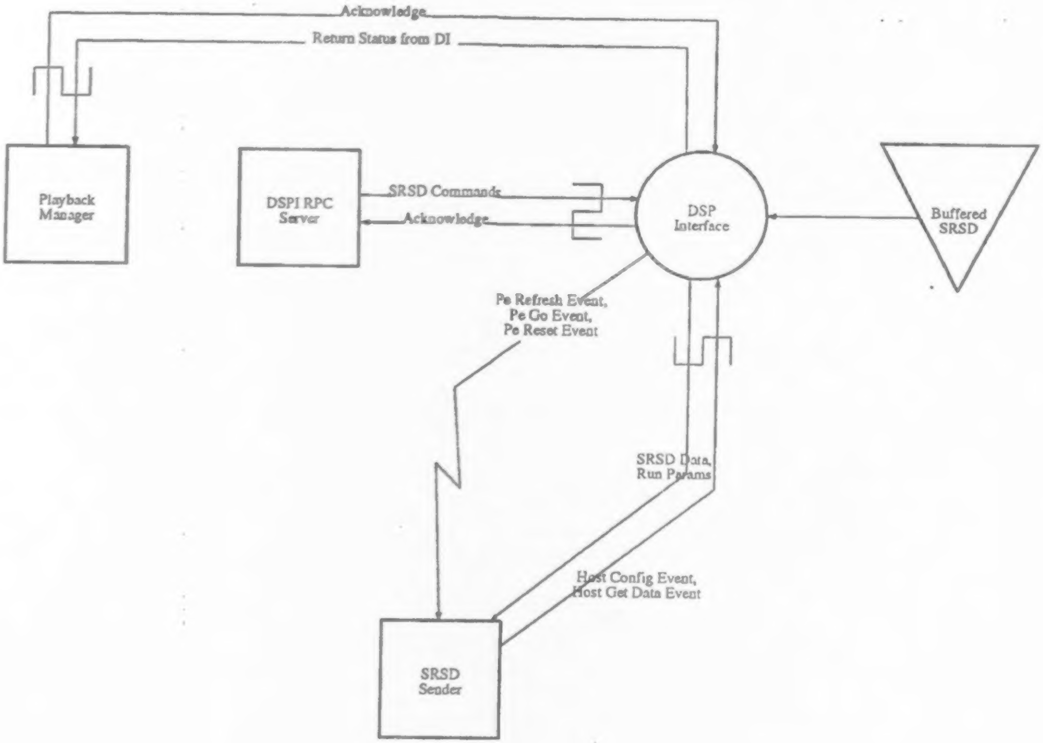

Acknowledge
Return Status from DI
Playback Manager
DSPI RPC Server
SRSD Commands
Acknowledge
DSP Interface
Buffered SRSD
Pe Refresh Event,
Pe Go Event,
Pe Reset Event
SRSD Data,
Run Params
Host Config Event,
Host Get Data Event
SRSD Sender

Figure B-10 DSP INTERFACE CSC/CSU CONTEXT DATA FLOW DIAGRAM

The objects identified in the High Level Data Flow Diagram flow from the CSC/CSU Context Diagram to maintain consistency.

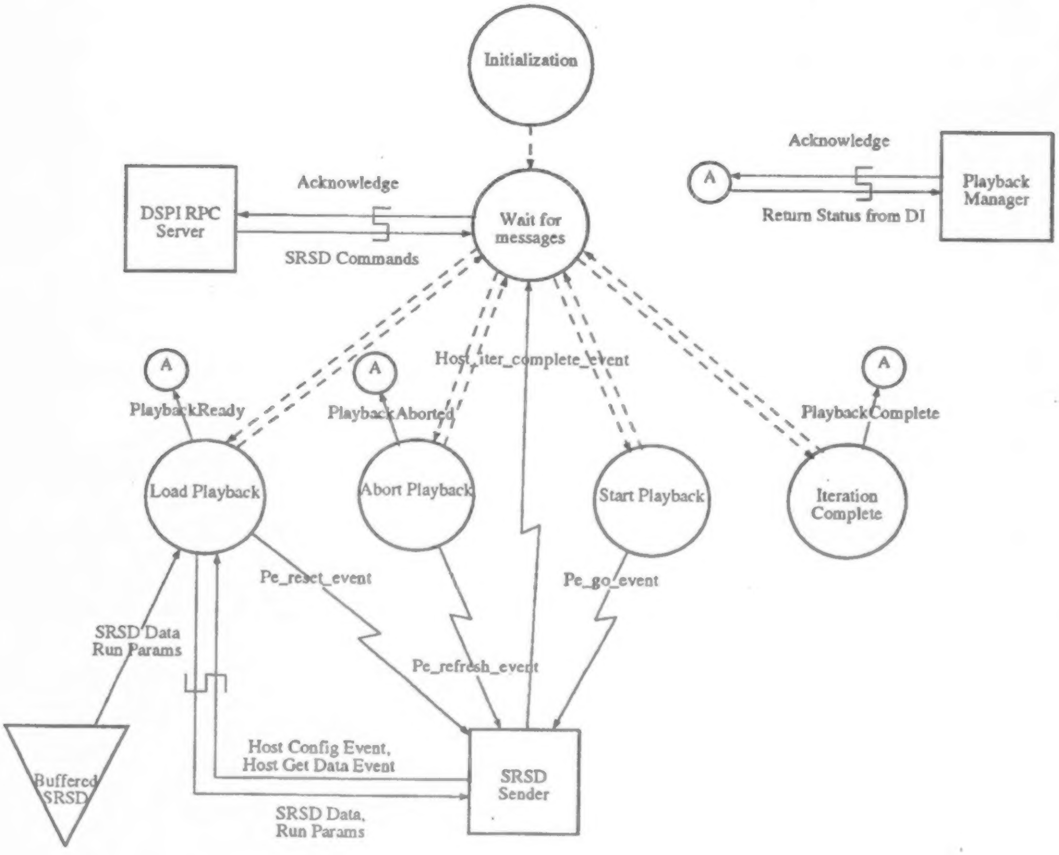

Initialization
DSPI RPC Server
Acknowledge
SRSD Commands
Wait for messages
A
Acknowledge
Return Status from DI
Playback Manager
Host_iter_complete_event
A
PlaybackReady
A
PlaybackAborted
A
PlaybackComplete
Load Playback
Abort Playback
Start Playback
Iteration Complete
Pe_reset_event
Pe_go_event
SRSD Data
Run Params
Pe_refresh_event
Host Config Event,
Host Get Data Event
SRSD Data,
Run Params
SRSD Sender
Buffered SRSD

Figure B-11 DSP INTERFACE CSC/CSU HIGH-LEVEL DATA FLOW DIAGRAM

B7.9.3 Module Design

B7.9.3.1 Design Logic

The design logic for each module shown in the DSP Interface CSU Module Invocation Tree is contained in Appendix 1 together with applicable design reports produced by CDADL.

Module Invocation Hierarchy

See Appendix E for invocation hierarchy.

B7.9.4 Module Test Plan

Table B-17 shows the module test plan for TDIT.

Module Name	Desk Inspection	Functional Test	Performance Analysis
f_tdit_dsp_interface	X		
f_tdit_initialize	X		
f_tdit_load_for_playback	X	X	
f_tdit_abort_playback	X		
f_tdit_start_playback	X	X	
f_tdit_time_iteration	X	X	

Table B-17 TDIT MODULE TEST PLAN

B7.9.5 DSP Interface CSU Test Plan

Objective 1. Verify that correct actions are performed for each SRSD Command.

Case 1:
Verify that the SRSD Command (LoadPlayback) is correctly received from the DSPI RPC Server, the Acknowledge is sent back and the LoadPlayback routine is correctly called, given a SRSD Command (LoadPlayback) from the DSPI RPC Server.

Case 2:
Verify that the SRSD Command (AbortPlayback) is correctly received from the DSPI RPC Server, the Acknowledge is sent back and the AbortPlayback routine is correctly called, given a SRSD Command (AbortPlayback) from the DSPI RPC Server.

Case 3:
Verify that the SRSD Command (StartPlayback) is correctly received from the DSPI RPC Server, the Acknowledge is sent back and the StartPlayback routine is correctly called, given a SRSD Command (StartPlayback) from the DSPI RPC Server.

Case 4:
Verify that the IterationComplete routine is correctly called, given a pe event flag (Host_iter_complete_event) from the SRSD Sender.

Objective 2. Verify the reading and writing of data.

Case 1: (LoadPlayback)
Verify that the PE_RESET_EVENT is sent to the SRSD Sender, the Host Config Event and the Host Get Data Event are correctly received from the SRSD Sender, the Run Params and SRSD Data are correctly sent to the SRSD Sender, and that the PlaybackReady is sent to the Playback Manager.

Case 2: (AbortPlayback)
Verify that the PE_REFRESH_EVENT is sent to the SRSD Sender, and the PlaybackAborted is sent to the Playback Manager.

Case 3: (StartPlayback)
Verify that the PE_GO_EVENT is sent to the SRSD Sender.

Case 4: (IterationComplete)
Verify that the PlaybackComplete is sent to the Playback Manager.

Objective 3. Verify that the DSP Interface handles error operations as designed.

B7.10 DSP Output CSU (TIOP)

B7.10.1 Introduction

The DSP Output CSU's (TIOP) main function is to playback simulated radar signal data (SRSD) to the DSP. It operates under the control of the DSP Interface CSC/CSU. It responds to control signals sent by DSP Interface that tell it to load SRSD data for a run, playback the data to the DSP, abort a run and so on. During the run for each iteration of the data, TIOP signals TDIT that it has completed the run. This allows DSP Interface to indicate the performance of the system.

B7.10.2 CSC/CSU Data Flow Diagrams

The ESP CSC/CSU Context Diagram is extracted from the ESP Inter-CSC/CSU Communication Diagram found in the ESP CSC Definition design note.

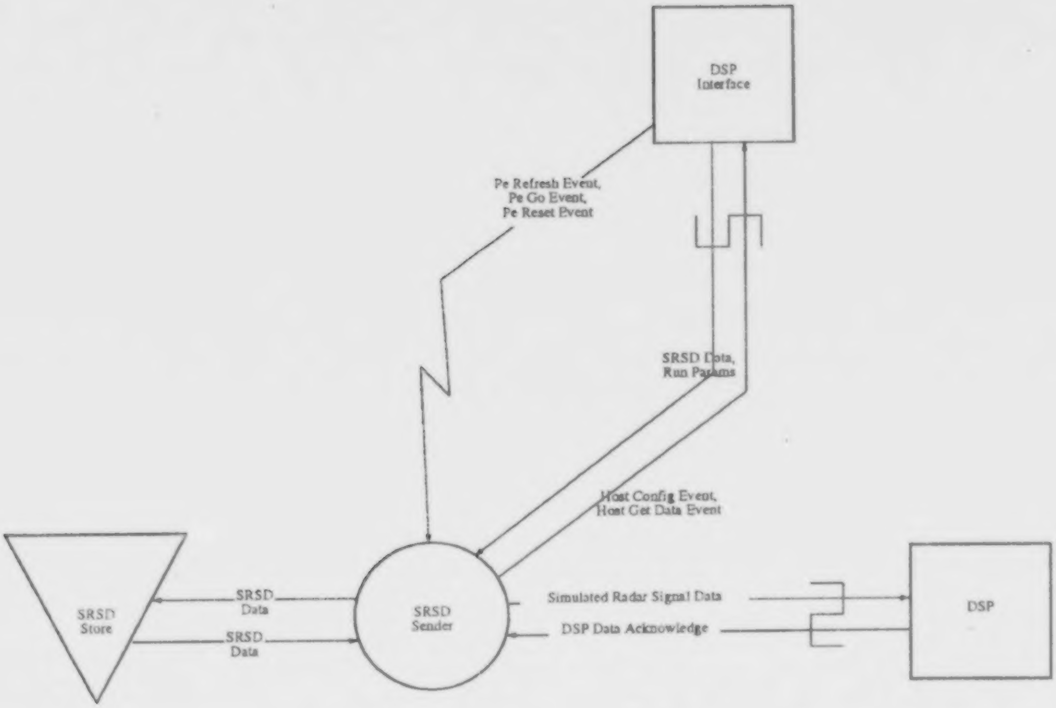

DSP
Interface
Pe Refresh Event,
Pe Go Event,
Pe Reset Event
SRSD Data,
Run Params
Host Config Event,
Host Get Data Event
SRSD
Store
SRSD
Data
SRSD
Data
SRSD
Sender
Simulated Radar Signal Data
DSP Data Acknowledge
DSP

Figure B-12 TIOP CSC/CSU CONTEXT DATA FLOW DIAGRAM

The objects identified in the High Level Data Flow Diagram flow from the CSC/CSU Context Diagram to maintain consistency.

B7.10.3 Module Design

B7.10.3.1 Design Logic

The design logic for each module shown in the TIOP CSU Module Invocation Tree is contained in Appendix E together with applicable design reports produced by CDADL.

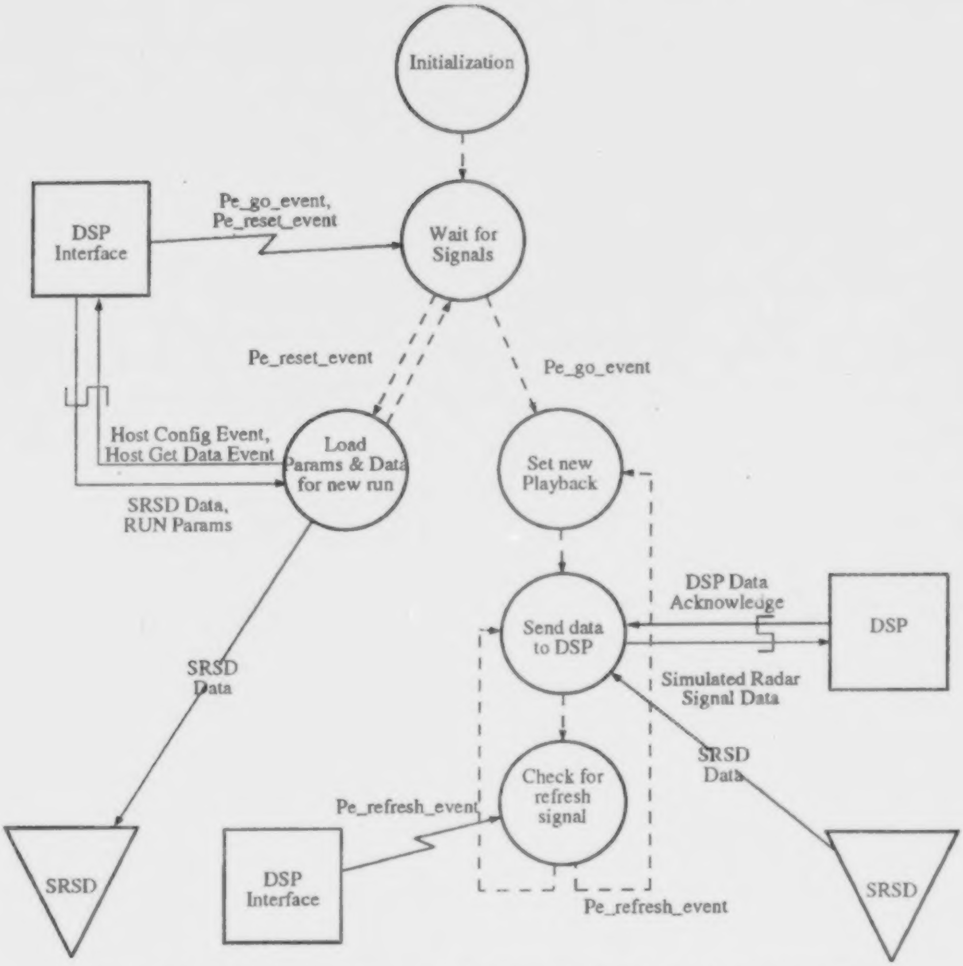

Initialization
DSP Interface
Pe_go_event, Pe_reset_event
Wait for Signals
Pe_reset_event
Pe_go_event
Host Config Event, Host Get Data Event
Load Params & Data for new run
SRSD Data, RUN Params
Set new Playback
Send data to DSP
DSP Data Acknowledge
DSP
Simulated Radar Signal Data
SRSD Data
SRSD Data
Check for refresh signal
Pe_refresh_event
DSP Interface
Pe_refresh_event
SRSD
SRSD

Figure B-13 TIOP CSC/CSU HIGH-LEVEL DATA FLOW DIAGRAM

Module Invocation Hierarchy

See Appendix E for invocation hierarchy.

B7.10.4 Module Test Plan

Table B-18 shows the module test plan for TIOP.

Module Name	Desk Inspection	Functional Test	Performance Analysis
f_tiop_srsd_sender	X		
f_tiop_playback	X	X	
f_tiop_load_new_run	X	X	

Table B-18 TIOP MODULE TEST PLAN

B7.10.5 TIOP CSU Test Plan

Objective 1. Verify that correct actions are performed for each Process Control Signal.

Case 1:
Verify that the Reset routine is correctly called, given a Processing Control Signal (PE_RESET_EVENT) from the DSP Interface.

Case 2:
Verify that the Playback routine is correctly called, given a Processing Control Signal (PE_GO_EVENT) from the DSP Interface.

Case 3:
Verify that the SRSD Sender aborts playing back radar data, given a Processing Control Signal (PE_REFRESH_EVENT) from the DSP Interface.

Objective 2. Verify the reading and writing of data.

Case 1: (RESET)
Verify that the Host Config Event and the Host Get Data Event are correctly sent to the DSP Interface, and that the Run Params and SRSD Data are correctly obtained from the DSP Interface, and put into the SRSD Store (TPM), given the PE_RESET_EVENT signal, the Run Params and SRSD Data from the DSP Interface.

Case 2: (GO)
Verify that the Iteration_count runs of SRSD Data are correctly retrieved from the SRSD Store (TPM) and sent to the DSP as given in the DSP-Testbed Interface Specification, given the PE_GO_EVENT signal.

Objective 3. Verify that the SRSD Sender handles error operations as designed.

B7.11 RPC Server CSU (TRSV)

B7.11.1 Introduction

The RPC Server CSC/CSU passes on commands it receives from Playback Manager in the form of clnt_calls and sends them on to DSP Interface CSC/CSU. It then waits for an Acknowledge from DSP Interface and passes this Acknowledge back to PlayBack Manager as an rpc reply.

B7.11.2 CSC/CSU Data Flow Diagrams

The ESP CSC/CSU Context Diagram is extracted from the ESP Inter-CSC/CSU Communication Diagram found in the ESP CSC Definition design note.

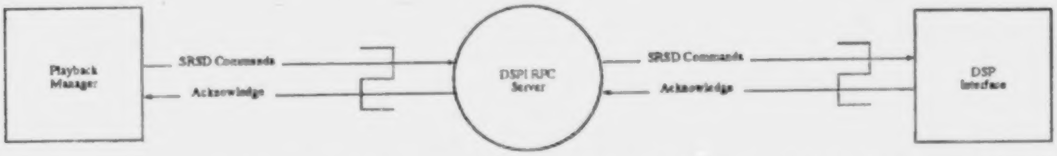

Playback Manager
SRSD Commands
Acknowledge
DSPI RPC Server
SRSD Commands
Acknowledge
DSP Interface

Figure B-14 TRSV CSC/CSU CONTEXT DATA FLOW DIAGRAM

The objects identified in the High Level Data Flow Diagram flow from the CSC/CSU Context Diagram to maintain consistency.

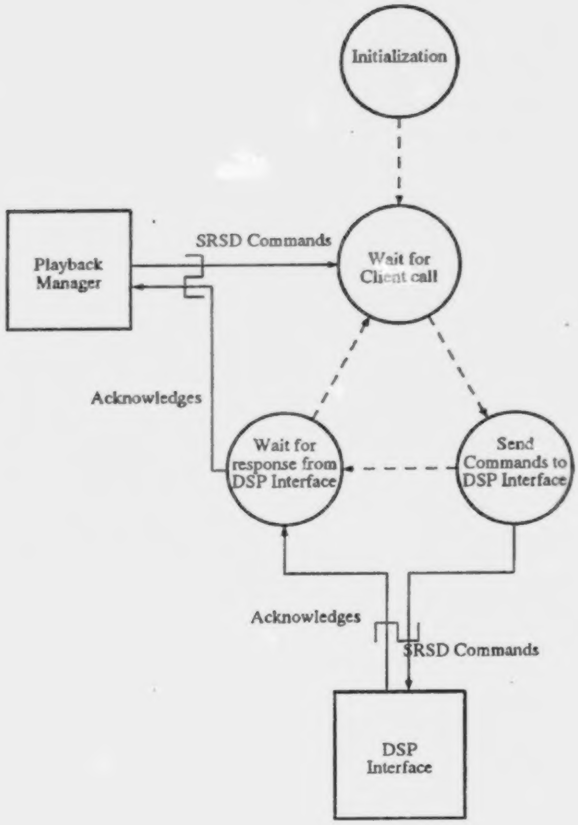

Initialization
Playback Manager
SRSD Commands
Wait for Client call
Acknowledges
Wait for response from DSP Interface
Send Commands to DSP Interface
Acknowledges
SRSD Commands
DSP Interface

Figure B-15 TRSV CSC/CSU HIGH-LEVEL DATA FLOW DIAGRAM

B7.11.3 Module Design

B7.11.3.1 Design Logic

The design logic for each module shown in the DSPI RPC Server CSU Module Invocation Tree is contained in Appendix E together with applicable design reports produced by CDADL.

```
Module Invocation Hierarchy

See Appendix E for invocation hierarchy.
```

B7.11.4 Module Test Plan

Table B-19 shows the module test plan for DSPI.

Module Name	Desk Inspection	Functional Test	Performance Analysis
f_trsv_dspi_rpc_server	X	X	

Table B-19 TRSV MODULE TEST PLAN

B7.11.5 RPC Server CSC Test Plan

Objective 1. Verify the receiving and sending of commands and acknowledges.

Case 1:
Verify that SRSD Commands are correctly received from the Playback Manager and sent to the DSP Interface, given SRSD Commands from the Playback Manager.

Case 2:
Verify that Acknowledges are correctly received from the DSP Interface and sent back to the Playback Manager, given the Acknowledge from the DSP Interface.

Objective 2. Verify that the DSPI RPC Server handles error operations as designed.

APPENDIX C

DETAILED DESIGN STANDARDS AND CONVENTIONS

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

Page

APPENDIX C DETAILED DESIGN STANDARDS AND CONVENTIONS C-1

C1 Purpose C-1
C2 Overview C-1
C3 Types of Files C-1

C3.1 Include File (.h) C-1
C3.2 Source File (.c) C-1
C3.3 Makefile (.mk) C-2
C3.4 Shell Script File (.sh) C-2

C4 CML Keyword Conventions C-2

C5 Package Model Standards and Conventions C-2

C5.1 Definitions C-2
C5.2 Files in a Package C-3

C6 File Naming Convention C-3

C6.1 Include File C-3
C6.2 Source File, Object File and Executable C-4
C6.3 Makefile and Library C-4

C7 Identifier Naming Conventions C-4

C8 PDL Conventions C-5

C8.1 PDL Guidelines C-5
C8.2 PDL Syntax C-5

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

DETAILED DESIGN STANDARDS AND CONVENTIONS

C1 Purpose

This technical note provides a summary of the coding standards as well as naming conventions to be followed in the detailed design and implementation phases, in order to make the SPD system consistent in a module's interface design as well as programming. The consistency in coding will make the program easier to understand and thus enhance the overall maintainability of the system. This technical note is based on Appendix A of the Software and Hardware Development Plan (Document R-3) and is considered as the extension of this appendix.

C2 Overview

Naming conventions are provided for four kinds of files in C: include file, source file, makefile and shell script file. The naming conventions of other file types, like data files, result files, and error files, will not be mentioned in this technote since they are provided in the data file subsystem description, Subsection A5.

C3 Types of Files

C3.1 Include File (.h)

It includes external interface to the packages: constants, data types, data structures and function specifications. The function specification consists of its name, formal parameter(s), and result(s).

C3.2 Source File (.c)

It includes the implementation software coded in C which might be export functions (callable by other packages) or local functions (accessible only by the same package). The export function's definitions must be declared in the exported include file.

C3.3 Makefile (.mk)

It includes the instructions for building the package library or executable using the UNIX make utility. The compilation or linking options and dependency of the packages (e.g. need other package' s includes or libraries) will be specified in this file.

C3.4 Shell Script File (.sh)

It uses UNIX shell commands to facilitate a sequence of run instructions in a physical file. It is normally provided to build the executables, or run unit testing for a package, or to invoke an executable which needs initialization of some system variables or user-entered parameters.

C4 CML Keyword Conventions

All source files, makefiles, shell script files to be placed by the librarian using the CML tool must have certain keywords embedded in them. The keywords used in CML is [@(#)]. These keywords must be present for not only the original version of the file but also every change submitted to the Configuration Management Coordinator (CMC).

Types of source files which require these keywords:

- Include file (.h)
- C source file (.c)
- Makefile (.mk)
- Shell Script file (.sh)
- ASCII files, ie: system configuration file.

Once files with this keyword have been submitted to the CML library and fetched out again, CML expands the keyword, replacing them with the file name, version and date submitted.

KeywordMeaningExample
[@(#)]file name, version & date@(#)

gives:"pkgxyzabc.c,1,92/12/18"

C5 Package Model Standards and Conventions

C5.1 Definitions

CSC: a Computer Software Component is a software development component, which contains one or more CSU(s). Each CSC must have a unique three-character name. The Testbed subsystem consists of two top level CSCs, the Controller (CLR) and Radar Signal Simulator (RSS).

CSU: a Computer Software Unit is a software development unit. Each CSU must have a unique four-character name. All CSUs under the CLR and RSS CSCs will be prefixed by "C" and "R" respectively, and followed by a three character acronym describing the CSU functionality.

Package: A package is equivalent to the CSU mentioned above.

A package can have many physical source files referred to as package unit or package element but only one export include file. The interface to the package is defined in the export include file, which contains declarations of constants, data types and functions provided by this package. Other packages using this package's service routines must include its export include file. It should be noted that if a package only defines data structure declaration and provides no service functions, then it can have only include file without having a source file.

Each package should have its own makefile. If a package contains a main program, then after running the makefile it will produce an executable. Otherwise, the source files are compiled and archived into a library file (.a). A package can generate more than one executable, ie: there will be more than one main program source file, main(), in one package.

Package Element: a single, named, independently compilable unit of code. A package can be divided into many source files or package elements, which is a collection of modules. The modules in a package element are bounded by cohesive functionality or data abstraction.

Module: a single, named, independently callable unit of code. It may be a main program, or a function in C.

C5.2 Files in a Package

Normally a package contains the following files:

a) one export include file (public) for the entire package;

b) zero or one local include file (private) for entire package;

c) zero or more source files (.c);

d) a makefile to compile all source files within a package. After running the makefile, it will produce executable(s) and/or generate a library (support services).

C6 FILE NAMING CONVENTIONS

C6.1 Include File

The general format of an include file name should be:

<package>_<keyword>.h

The <package> is the four-character package name in lower-case. The <key word> may be "intf" or "priv". The "intf" refers to the export include file and "priv" refers to the package's private include file which declares global definitions for all package units within a package.

C6.2 Source File, Object File and Executable

The general format of a source file, object file and executable file name should be:

<package>_<name>.c- source file
<package>_<name>.o- object file
<package>_<name>- executable

The <package> is a four-character lower-case package name. The <name> is the package unit name, up to 16-letters long in lower case.

C6.3 Makefile and Library

The general format of a makefile and library file name should be:

<package>.mk- makefile
<package>.a- library

The <package> a four-character lower-case package name.

C7 Identifier Naming Conventions

The general format of an exported identifier in a C program should be:

X_<package>_name

where:
X:Entity identification code. The following standard codes will be used:

constant	c
types	t
exception	x
function names	f
interrupt handler	i
macros	m
status codes	s
global variables	g

<package>:	four-character package name in lower-case.
name:	free-format string but it should be understandable, significant, and meaningful.

Examples:

t_pkg_NCI_waveform	:	NCI waveform structure
s_pkg_fatal	:	returned status
g_pkg_detection_array	:	global array for radar detections
f_pkg_derive_sat_loc		:function to derive satellite location

These external identifiers are declared in the corresponding include file for a package. However, we relax this rule for internal identifiers which are not referenced by other source files. It means that local constants, local data types, local variables, local macros, static functions etc. do not need to have the X_<package> prefix.

C8 PDL Conventions

PDL is a language for communicating designs between designers and between designers and programmers. It is used to provide a high-level definition of a design which can be reviewed for completeness and accuracy. Therefore, the designer should document a design completely, clearly and unambiguously in PDL. Style and layout play an important part in the ease of understanding of a design.

C8.1 PDL Guidelines

PDL is used during detailed design to describe and document the algorithms to be implemented in modules. The purpose of the PDL is to provide a simplified description of the design of the module and prevent implementation details from obscuring this design description.

The major reason for using PDL as an intermediate step between preliminary design descriptions and the coding phase is to permit meaningful review of the module's design. Design changes are much cheaper and easier to make at this stage than at any later stage.

The PDL syntax chosen reflects a high level of abstraction, using a logical level of description. It is tailored to provide an easily understandable description of what is being done while simultaneously establishing the framework and bounds of how it will be done. Although PDL indicates the functions, algorithms and design of a module and does not express implementation details, it does mirror the implementation environment to the extent that the coding activity consists primarily of implementing the PDL design.

The PDL description describes the design with simplicity and clarity and every effort is made to be concise and unambiguous. This generally means using a simplified and formalized vocabulary with simplified grammatical rules. Thus compound or complex sentences with indirect objects or passive voice is avoided.

C8.2 PDL Syntax

The PDL uses fairly free-form English descriptions mixed with flow of control statements. The PDL syntax can use anything like in C, ie: assignment, control flow, case statement etc. Typically, one line of PDL in a detailed design corresponds to five to ten lines of the eventual source code. Control flow statements in PDL will not require compilation, therefore, syntax can be de-emphasised. It is more important to provide a complete definition than it is to have the correct PDL syntax.

C8.2.1 Function/Main Program Declaration

```
BEGIN function_or_program_name (brief description)
  <statement>
END function_or_program_name
```

C8.2.2 Expressions and Operators

Expressions are formed by applying operators to variables, constants and functions. The allowed operators are: **, *, /, MOD, REM, DIV, +, -, =, <>, <, <=, AND, OR, NOT.

Example:

```
AJ = REM(BI+J,NAUX) + 1
IF (NB = 2) AND (NAUX = 2) THEN
```

C8.2.3 Function Calls

```
USE <function name> TO <perform process>
```

C8.2.4 IF Statement

```
IF <condition> THEN
  <statement>
ENDIF
```

or

```
IF <condition 1> THEN
  <statement sequence 1>
ELSEIF < condition 2> THEN
  <statement sequence 2>
ELSE
  <statement sequence n>
ENDIF
```

Example:

```
IF unlink file returns unsuccessfully THEN
  USE msglog TO issue an error message
ENDIF
```

C8.2.5 CASE Statement

```
CASE <expression> DO
  <case 1>:  <statement sequence 1>
  <case 2>:  <statement sequence 2>
  default:   <statement sequence n>
ENDCASE
```

Example:

```
CASE process state DO
    idle:
            clear statistics, clear timer
    ready:
            format message
            USE xxx TO send message
    default:
            do nothing
ENDCASE
```

C8.2.6 FOR Loop or WHILE Loop

```
WHILE <condition> DO
  <statement sequence>
ENDWHILE

  or

FOR <variable>  IN <range>DO                FOR <statement> DO
  <statement sequence>                        <statement sequence>
ENDFOR                                      ENDFOR
```

Example:

```
WHILE (not end of file) OR (not shutdown) DO
  send a line to printer
  read next line from file
ENDWHILE

FOR I IN 1, number of blankers DO
  BI = I * DAUX
ENDFOR

FOR each blanker DO
  USE xxx TO derive parameters
ENDFOR
```

APPENDIX D

CONTROLLER PDL

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

Page

APPENDIX D CONTROLLER PDL **D-1**

D1 CONTROLLER PACKAGES D-1

D1.1 Satellite/Radar HMI (CRDH) D-1
D1.2 Environment HMI (CENH) D-23
D1.3 Simulation Run HMI (CSDH) D-71
D1.4 RSS Generation Interface (CRGI) D-85
D1.5 RSS Playback Interface (CRPI) D-95
D1.6 DSP Simulation Interface (CDSI) D-105
D1.7 Data Analysis (CDAT) D-107
D1.8 Satellite/Radar Definition (CRAD) D-161
D1.9 Environment Definition (CEND) D-181
D1.10 Simulation Run Definition (CSID) D-193
D1.11 Simulation Products (CSIP) D-203
D1.12 Signal Generation Products (CGEP) D-205
D1.13 Control Generation HMI (CGEH) D-217
D1.14 Generation Server (CGET) D-229
D1.15 Control Simulation HMI (CSIH) D-239
D1.16 Simulation Server (CSIT) D-251
D1.17 Data Validation (CVAL) D-263
D1.18 Error Logging (CERR) D-363
D1.19 System Services (CSYS) D-373
D1.20 User Interface Services (CUIS) D-387

APPENDIX D

CONTROLLER PDL

D1 CONTROLLER PACKAGES

D1.1 Satellite/Radar HMI (CRDH)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      crdh_priv.h - Satellite/Radar Definition HMI
                              private include file

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file declares constants (defaults and ranges) for
                a Satellite/Radar Definition.

Designer:       Stephen Tran

Change History:

    SCR    Date        Author   Change made
    ---  -----------   ------  ----------------------------------------
    n/a  26-Apr-1993    cst   Initial Release

===========================================================================*/

#ifndef crdh_priv_h
#define crdh_priv_h

/*-------------------------------------------------------------------------- *
 *                                                                           *
 *    DEFINE CONSTANTS                                                       *
 *                                                                           *
 *---------------------------------------------------------------------------*/
/* frequency band                       SAT HMI 1.1.2 */
#define c_crdh_dfl_RRA_L_freq_mid       1.3     /* GHz */
#define c_crdh_dfl_RRA_S_freq_mid       3.0     /* GHz */
#define c_crdh_dfl_CFA_L_bandwidth      200.0   /* GHz */
#define c_crdh_dfl_CFA_S_bandwidth      400.0   /* GHz */
#define c_crdh_min_freq_mid             1       /* GHz */
#define c_crdh_max_freq_mid             2       /* GHz */
#define c_crdh_min_bandwidth            0       /* MHz */
#define c_crdh_max_bandwidth            2000    /* MHz */

/* altitude                             SAT HMI 2.1.1 */
#define c_crdh_dfl_altitude             850.0   /* km */
#define c_crdh_min_altitude             500.0   /* km */
#define c_crdh_max_altitude             10000.0 /* km */

/* incl_angle                           SAT HMI 2.1.2 */
#define c_crdh_dfl_incl_angle           75.0    /* degrees */
#define c_crdh_min_incl_angle           0.0     /* degrees */
#define c_crdh_max_incl_angle           90.0    /* degrees */

/* longitude */
#define c_crdh_dfl_longitude            0.0     /* degrees */
#define c_crdh_min_longitude            -180.0  /* degrees */
#define c_crdh_max_longitude            180.0   /* degrees */

/* latitude */
#define c_crdh_dfl_latitude             0.0     /* degrees */
#define c_crdh_min_latitude             -90.0   /* degrees */
#define c_crdh_max_latitude             90.0    /* degrees */

/* azimuth */
```

```
#define c_crdh_dfl_azimuth              0.0     /* degrees */
#define c_crdh_min_azimuth              0.0     /* degrees */
#define c_crdh_max_azimuth              360.0   /* degrees */
#define c_crdh_min_tracking_azimuth     -90.0   /* degrees */
#define c_crdh_max_tracking_azimuth     90.0    /* degrees */

/* grazing */
#define c_crdh_dfl_grazing              0.0     /* degrees */
#define c_crdh_min_grazing              0.0     /* degrees */
#define c_crdh_max_grazing              90.0    /* degrees */

/* roll angle */
#define c_crdh_dfl_roll                 0.0     /* degrees */
#define c_crdh_min_roll                 -90.0   /* degrees */
#define c_crdh_max_roll                 90.0    /* degrees */

/* pitch angle */
#define c_crdh_dfl_pitch                0.0     /* degrees */
#define c_crdh_min_pitch                -90.0   /* degrees */
#define c_crdh_max_pitch                90.0    /* degrees */

/* yaw angle */
#define c_crdh_dfl_yaw                  0.0     /* degrees */
#define c_crdh_min_yaw                  -90.0   /* degrees */
#define c_crdh_max_yaw                  90.0    /* degrees */

/* elevation angle */
#define c_crdh_dfl_elevation            61.8    /* degrees */
#define c_crdh_min_elevation            0.0     /* degrees */
#define c_crdh_max_elevation            90.0    /* degrees */
#define c_crdh_min_tracking_elevation   -90.0   /* degrees */
#define c_crdh_max_tracking_elevation   90.0    /* degrees */

/* horn gain                            SAT HMI 2.2.4.1, 2.3.9.1, 2.4.2.1 */
#define c_crdh_dfl_horn_gain            10.0    /* dBi */
#define c_crdh_min_horn_gain            0.0     /* dBi */
#define c_crdh_max_horn_gain            20.0    /* dBi */

/*-------------------------------------------------------------------------
   fixed default and fixed range for CFA Antenna
  -------------------------------------------------------------------------*/
/* CFA aperture (along motion X)        SAT HMI 2.2.1 */
#define c_crdh_dfl_CFA_aper_x           22.0    /* metres */
#define c_crdh_min_CFA_aper_x           0.0     /* metres */
#define c_crdh_max_CFA_aper_x           100.0   /* metres */

/* CFA aperture (along motion Y)        SAT HMI 2.2.1 */
#define c_crdh_dfl_CFA_aper_y           6.0     /* metres */
#define c_crdh_min_CFA_aper_y           0.0     /* metres */
#define c_crdh_max_CFA_aper_y           100.0   /* metres */

/* grid spacing dX & dY                 SAT HMI 2.2.3 */
#define c_crdh_dfl_grid_spacing         0.115   /* metres */
#define c_crdh_min_grid_spacing         0.0     /* metres */
#define c_crdh_max_grid_spacing         0.3     /* metres */

/* PRI multipler                        SAT HMI 2.2.6.2 */
#define c_crdh_dfl_PRI_multipler        0.0     /* integer */
#define c_crdh_min_PRI_multipler        -180.0  /* integer */
#define c_crdh_max_PRI_multipler        180.0   /* integer */

/* displacement from aperture           SAT HMI 2.2.6.3 */
#define c_crdh_dfl_disp                 0.0     /* metres */
#define c_crdh_min_disp                 -50.0   /* metres */
#define c_crdh_max_disp                 50.0    /* metres */

/* near_SL                              SAT HMI 2.2.7.1, 2.2.8.1 */
#define c_crdh_dfl_near_SL              -40.0   /* dB */
#define c_crdh_min_near_SL              -20.0   /* dB */
#define c_crdh_max_near_SL              -50.0   /* dB */
```

```

/*------------------------------------------------------------------------
   fixed default and fixed range for RRA Antenna
  -------------------------------------------------------------------------*/
/* Main reflector diameter                 SAT HMI 2.3.1 */
#define c_crdh_dfl_main_diameter           25.0     /* metres */
#define c_crdh_min_main_diameter           0.0      /* metres */
#define c_crdh_max_main_diameter           100.0    /* metres */

/* Main reflector focal length             SAT HMI 2.3.2 */
#define c_crdh_dfl_main_focal_len          12.5     /* metres */
#define c_crdh_min_main_focal_len          0.0      /* metres */
#define c_crdh_max_main_focal_len          100.0    /* metres */

/* Subreflector diameter                   SAT HMI 2.3.3 */
#define c_crdh_dfl_sub_diameter            2.5      /* metres */
#define c_crdh_min_sub_diameter            0.0      /* metres */
#define c_crdh_max_sub_diameter            100.0    /* metres */

/* Subreflector focal length               SAT HMI 2.3.4 */
#define c_crdh_dfl_sub_focal_len           1.25     /* metres */
#define c_crdh_min_sub_focal_len           0.0      /* metres */
#define c_crdh_max_sub_focal_len           100.0    /* metres */

/* Feed diameter                           SAT HMI 2.3.5 */
#define c_crdh_dfl_feed_diameter           2.5      /* metres */
#define c_crdh_min_feed_diameter           0.0      /* metres */
#define c_crdh_max_feed_diameter           100.0    /* metres */

/* Feed axial position                     SAT HMI 2.3.6 */
#define c_crdh_dfl_feed_axial_pos          0.0      /* metres */
#define c_crdh_min_feed_axial_pos          0.0      /* metres */
#define c_crdh_max_feed_axial_pos          100.0    /* metres */

/* Feed grid spacing dX & dY               SAT HMI 2.3.8 */
#define c_crdh_dfl_feed_grid_spacing       0.08     /* metres */
#define c_crdh_min_feed_grid_spacing       0.0      /* metres */
#define c_crdh_max_feed_grid_spacing       0.3      /* metres */

/* RMS aperture distribution error         SAT HMI 2.2.9 & 2.3.12 */
#define c_crdh_dfl_RMS_aper_dist_err       0.1      /* float */
#define c_crdh_min_RMS_aper_dist_err       0.0      /* float */
#define c_crdh_max_RMS_aper_dist_err       1.0      /* float */

/* Rotation rate                           SAT HMI 2.3.13 */
#define c_crdh_dfl_rotation_rate           2.5      /* rpm */
#define c_crdh_min_rotation_rate           -6.0     /* rpm */
#define c_crdh_max_rotation_rate           6.0      /* rpm */

/*------------------------------------------------------------------------
   fixed default and fixed range for Auxiliary Antenna
  -------------------------------------------------------------------------*/
/* Number of auxiliary antennae            SAT HMI 2.4 */
#define c_crdh_dfl_aux_count               6        /* integer */
#define c_crdh_min_aux_count               0        /* integer */
#define c_crdh_max_aux_count               16       /* integer */

/* Auxiliary phase center loc (X, Y, Z)    SAT HMI 2.4.3 */
#define c_crdh_dfl_ph_ctr_loc              0.0      /* metres */
#define c_crdh_min_ph_ctr_loc              -100.0   /* metres */
#define c_crdh_max_ph_ctr_loc              100.0    /* metres */

/* tap delay                               SAT HMI 2.4.5.1 */
#define c_crdh_dfl_Tmax_1                  30.0     /* ns */
#define c_crdh_dfl_Tmax_2                  45.0     /* ns */
#define c_crdh_dfl_Tmax_3                  60.0     /* ns */
#define c_crdh_min_tap_delay               1.0      /* ns */
#define c_crdh_max_tap_delay               300.0    /* ns */

/*------------------------------------------------------------------------
```

```
     fixed default and fixed range for regular array auxiliary
   ---------------------------------------------------------------------------*/
 /* regular array aperture (along motion X and Y) SAT HMI 2.4.6.3 */
 #define c_crdh_dfl_reg_aper               0.0     /* metres */
 #define c_crdh_min_reg_aper               0.0     /* metres */
 #define c_crdh_max_reg_aper               100.0   /* metres */

 /* regular array grid spacing dX & dY    SAT HMI 2.4.6.5 */
 #define c_crdh_dfl_reg_grid_spacing       0.08    /* metres */
 #define c_crdh_min_reg_grid_spacing       0.0     /* metres */
 #define c_crdh_max_reg_grid_spacing       0.3     /* metres */

 /* regular array near_SL                 SAT HMI 2.4.6.6.1 */
 #define c_crdh_dfl_reg_near_SL            -30.0   /* dB */
 #define c_crdh_min_reg_near_SL            -20.0   /* dB */
 #define c_crdh_max_reg_near_SL            -50.0   /* dB */

 /*--------------------------------------------------------------------------
    fixed default and fixed range for arbitrary array auxiliary
   ---------------------------------------------------------------------------*/
 /* arbitrary array location (X, Y, Z)    SAT HMI 2.4.7.2.1 */
 #define c_crdh_dfl_arb_aper               0.0     /* metres */
 #define c_crdh_min_arb_aper               -100.0  /* metres */
 #define c_crdh_max_arb_aper               100.0   /* metres */

 /* fixed  phase shift                    SAT HMI 2.4.7.2.2 */
 #define c_crdh_dfl_ph_shift_odd           0.0     /* degrees */
 #define c_crdh_dfl_ph_shift_even          180.0   /* degrees */
 #define c_crdh_min_ph_shift               -180.0  /* degrees */
 #define c_crdh_max_ph_shift               -180.0  /* degrees */

 /* fixed amplitude weight                SAT HMI 2.4.7.2.3 */
 #define c_crdh_dfl_amp_weight             1.0     /* float */

 /*--------------------------------------------------------------------------
    fixed default and fixed range for Receive Front End
   ---------------------------------------------------------------------------*/
 /* bandwidth                             SAT HMI 2.5.1 */
 #define c_crdh_dfl_rxfe_bandwidth         1.0     /* MHz */
 #define c_crdh_min_rxfe_bandwidth         0.1     /* MHz */
 #define c_crdh_max_rxfe_bandwidth         6.0     /* MHz */

 /* sampling rate                         SAT HMI 2.5.2 */
 #define c_crdh_dfl_rxfe_sampling_rate     1.5     /* MHz */
 #define c_crdh_min_rxfe_sampling_rate     0.1     /* MHz */
 #define c_crdh_max_rxfe_sampling_rate     9.0     /* MHz */

 /* main beam delay                       SAT HMI 2.5.6 */
 #define c_crdh_dfl_main_beam_delay        0.0     /* ns */
 #define c_crdh_min_main_beam_delay        0.0     /* ns */
 #define c_crdh_max_main_beam_delay        300.0   /* ns */

 /* A/D dynamic range bits                SAT HMI ??? */
 #define c_crdh_dfl_AD_dynamic_rng         8       /* integer */   /* ??? */
 #define c_crdh_min_AD_dynamic_rng         8       /* integer */
 #define c_crdh_max_AD_dynamic_rng         16      /* integer */

 /*--------------------------------------------------------------------------
    fixed default and fixed range for Analog Effects
   ---------------------------------------------------------------------------*/
 /* RX Noise temperature                  SAT HMI 2.5.5.1 */
 #define c_crdh_dfl_noise_temp             25.0    /* Kelvin */
 #define c_crdh_min_noise_temp             0.0     /* kelvin */
 #define c_crdh_max_noise_temp             10000.0 /* Kelvin */

 /* IF & Equalizer character              SAT HMI 2.5.5.3 */
 #define c_crdh_dfl_IF_amplitude           -50.0   /* dB */
 #define c_crdh_min_IF_amplitude           -100.0  /* dB */
 #define c_crdh_max_IF_amplitude           -30.0   /* dB */

```

```
#define c_crdh_dfl_IF_phase             0.1       /* degrees */
#define c_crdh_min_IF_phase             0.0       /* degrees */
#define c_crdh_max_IF_phase             10.0      /* degrees */

#define c_crdh_dfl_IF_ripple            6         /* integer */
#define c_crdh_min_IF_ripple            1         /* integer */
#define c_crdh_max_IF_ripple            20        /* integer */

/* I/Q Mismatch character               SAT HMI 2.5.5.5 */
#define c_crdh_dfl_IQ_amplitude         0.0       /* dB */
#define c_crdh_min_IQ_amplitude         0.0       /* dB */
#define c_crdh_max_IQ_amplitude         -80.0     /* dB */

#define c_crdh_dfl_IQ_phase             0.0       /* degrees */
#define c_crdh_min_IQ_phase             -90.0     /* degrees */
#define c_crdh_max_IQ_phase             90.0      /* degrees */

#define c_crdh_dfl_DC_offset_I          0.0       /* bits */
#define c_crdh_min_DC_offset_I          -5.0      /* bits */
#define c_crdh_max_DC_offset_I          5.0       /* bits */

#define c_crdh_dfl_DC_offset_Q          0.0       /* bits */
#define c_crdh_min_DC_offset_Q          -5.0      /* bits */
#define c_crdh_max_DC_offset_Q          5.0       /* bits */

/*--------------------------------------------------------------------------
   fixed default and fixed range for Multipath Scatters
  --------------------------------------------------------------------------*/
/* position (x, y, z)                   SAT HMI 2.6.1 */
#define c_crdh_dfl_ms_position          0.0       /* metres */
#define c_crdh_min_ms_position          -50.0     /* meters */
#define c_crdh_max_ms_position          50.0      /* meters */

/* average reflectivity                 SAT HMI 2.6.2 */
#define c_crdh_dfl_avg_reflectivity     -40.0     /* dB */
#define c_crdh_min_avg_reflectivity     -80.0     /* dB */
#define c_crdh_max_avg_reflectivity     0.0       /* dB */

/* average lobe width                   SAT HMI 2.6.3 */
#define c_crdh_dfl_avg_lobe_width       10.0      /* degrees */
#define c_crdh_min_avg_lobe_width       0.0       /* degrees */
#define c_crdh_max_avg_lobe_width       -90.0     /* degrees */

/*--------------------------------------------------------------------------
   fixed default and fixed range for Dwell Interval
  --------------------------------------------------------------------------*/
/* beam location azimuth and grazing angles     SAT HMI 3.1.1 */
#define c_crdh_dfl_beam_azimuth                 90.0      /* degrees */
#define c_crdh_dfl_beam_grazing                 6 .0      /* degrees */

/* pulse duration                       SAT HMI 3.2.1 */
#define c_crdh_dfl_RRA_exp_pulse_dur    6.667     /* micro seconds */
#define c_crdh_dfl_CFA_exp_pulse_dur    14.667    /* micro seconds */
#define c_crdh_min_exp_pulse_dur        0.0       /* micro seconds */
#define c_crdh_max_exp_pulse_dur        1000.0    /* micro seconds */

/* compressed pulse width               SAT HMI 3.2.2 */
#define c_crdh_dfl_compr_pulse_width    0.667.0 /* micro seconds */
#define c_crdh_min_compr_pulse_width    0.0       /* micro seconds */
#define c_crdh_max_compr_pulse_width    1000.0    /* micro seconds */

/* peak pulse power                     SAT HMI 3.2.4 */
#define c_crdh_dfl_RRA_pulse_peak_power 5e+3      /* Watts */
#define c_crdh_dfl_CFA_pulse_peak_power 5e+4      /* Watts */
#define c_crdh_min_pulse_peak_power     0.0       /* Wattss */
#define c_crdh_max_pulse_peak_powerr    1e+6      /* Wattss */

/*--------------------------------------------------------------------------
   fixed default and fixed range for NCI Interval
  --------------------------------------------------------------------------*/
```

```
/* samples per PRI                          SAT HMI 4.1.1 */
#define c_crdh_dfl_RRA_smp_per_PRI_1     93       /* integer */
#define c_crdh_dfl_RRA_smp_per_PRI_2     97       /* integer */
#define c_crdh_dfl_RRA_smp_per_PRI_3     101      /* integer */
#define c_crdh_dfl_RRA_smp_per_PRI_4     95       /* integer */
#define c_crdh_dfl_RRA_smp_per_PRI_5     103      /* integer */
#define c_crdh_dfl_RRA_smp_per_PRI_6     94       /* integer */
#define c_crdh_dfl_RRA_smp_per_PRI_7     107      /* integer */
#define c_crdh_dfl_RRA_smp_per_PRI_8     91       /* integer */
#define c_crdh_dfl_RRA_smp_per_PRI_9     109      /* integer */
#define c_crdh_dfl_RRA_smp_per_PRI_10    89       /* integer */
#define c_crdh_dfl_RRA_smp_per_PRI_11    113      /* integer */
#define c_crdh_dfl_RRA_smp_per_PRI_12    83       /* integer */

#define c_crdh_dfl_CFA_smp_per_PRI_1     217      /* integer */
#define c_crdh_dfl_CFA_smp_per_PRI_2     221      /* integer */
#define c_crdh_dfl_CFA_smp_per_PRI_3     225      /* integer */
#define c_crdh_dfl_CFA_smp_per_PRI_4     223      /* integer */
#define c_crdh_dfl_CFA_smp_per_PRI_5     218      /* integer */
#define c_crdh_dfl_CFA_smp_per_PRI_6     211      /* integer */
#define c_crdh_dfl_CFA_smp_per_PRI_7     209      /* integer */
#define c_crdh_dfl_CFA_smp_per_PRI_8     227      /* integer */
#define c_crdh_dfl_CFA_smp_per_PRI_9     229      /* integer */
#define c_crdh_dfl_CFA_smp_per_PRI_10    233      /* integer */
#define c_crdh_dfl_CFA_smp_per_PRI_11    239      /* integer */
#define c_crdh_dfl_CFA_smp_per_PRI_12    241      /* integer */

#define c_crdh_min_CFA_smp_per_PRI       10       /* integer */
#define c_crdh_max_CFA_smp_per_PRI       1024     /* integer */

/* pulses per CPI                           SAT HMI 4.1.2 */
#define c_crdh_dfl_RRA_pul_per_CPI_1     80       /* integer */
#define c_crdh_dfl_RRA_pul_per_CPI_2     77       /* integer */
#define c_crdh_dfl_RRA_pul_per_CPI_3     74       /* integer */
#define c_crdh_dfl_RRA_pul_per_CPI_4     78       /* integer */
#define c_crdh_dfl_RRA_pul_per_CPI_5     72       /* integer */
#define c_crdh_dfl_RRA_pul_per_CPI_6     79       /* integer */
#define c_crdh_dfl_RRA_pul_per_CPI_7     70       /* integer */
#define c_crdh_dfl_RRA_pul_per_CPI_8     82       /* integer */
#define c_crdh_dfl_RRA_pul_per_CPI_9     68       /* integer */
#define c_crdh_dfl_RRA_pul_per_CPI_10    84       /* integer */
#define c_crdh_dfl_RRA_pul_per_CPI_11    66       /* integer */
#define c_crdh_dfl_RRA_pul_per_CPI_12    90       /* integer */

#define c_crdh_dfl_CFA_pul_per_CPI_1     315      /* integer */
#define c_crdh_dfl_CFA_pul_per_CPI_2     309      /* integer */
#define c_crdh_dfl_CFA_pul_per_CPI_3     304      /* integer */
#define c_crdh_dfl_CFA_pul_per_CPI_4     306      /* integer */
#define c_crdh_dfl_CFA_pul_per_CPI_5     314      /* integer */
#define c_crdh_dfl_CFA_pul_per_CPI_6     324      /* integer */
#define c_crdh_dfl_CFA_pul_per_CPI_7     327      /* integer */
#define c_crdh_dfl_CFA_pul_per_CPI_8     301      /* integer */
#define c_crdh_dfl_CFA_pul_per_CPI_9     298      /* integer */
#define c_crdh_dfl_CFA_pul_per_CPI_10    293      /* integer */
#define c_crdh_dfl_CFA_pul_per_CPI_11    286      /* integer */
#define c_crdh_dfl_CFA_pul_per_CPI_12    284      /* integer */

#define c_crdh_min_CFA_pul_per_CPI       1        /* integer */
#define c_crdh_max_CFA_pul_per_CPI       1500     /* integer */

/*-----------------------------------------------------------------------
   fixed default and fixed range for CPI Interval
  -----------------------------------------------------------------------*/
/* RF frequency                            SAT HMI 5.1.1 */
#define c_crdh_dfl_RRA_RF_freq_1        2.8      /* GHz */
#define c_crdh_dfl_RRA_RF_freq_2        2.93     /* GHz */
#define c_crdh_dfl_RRA_RF_freq_3        3.07     /* GHz */
#define c_crdh_dfl_RRA_RF_freq_4        3.2      /* GHz */

#define c_crdh_dfl_CFA_RF_freq_1        1.2      /* GHz */
```

```
#define c_crdh_dfl_CFA_RF_freq_2         1.24     /* GHz */
#define c_crdh_dfl_CFA_RF_freq_3         1.28     /* GHz */
#define c_crdh_dfl_CFA_RF_freq_4         1.32     /* GHz */
#define c_crdh_dfl_CFA_RF_freq_5         1.36     /* GHz */
#define c_crdh_dfl_CFA_RF_freq_6         1.4      /* GHz */

#define c_crdh_min_CFA_RF_freq           1.0      /* GHz */
#define c_crdh_max_CFA_RF_freq           4.0      /* GHz */

/*-----------------------------------------------------------------------------
   fixed default and fixed range for data reduction
  ----------------------------------------------------------------------------*/
/* Time decimation                       SAT HMI 6.12.1 */
#define c_crdh_dfl_time_decim_start      1        /* integer */
#define c_crdh_dfl_time_decim_stop       64       /* integer */
#define c_crdh_dfl_time_decim_step       1        /* integer */

#define c_crdh_min_time_decim_start      1        /* integer */
#define c_crdh_min_time_decim_stop       1        /* integer */
#define c_crdh_min_time_decim_step       1        /* integer */

#define c_crdh_max_time_decim_start      1500     /* integer */
#define c_crdh_max_time_decim_stop       1500     /* integer */
#define c_crdh_max_time_decim_step       1499     /* integer */

/* Range decimation                      SAT HMI 6.12.2 */
#define c_crdh_dfl_range_decim_start     1.0      /* float */
#define c_crdh_dfl_range_decim_stop      64.0     /* float */
#define c_crdh_dfl_range_decim_step      1.0      /* float */

#define c_crdh_min_range_decim_start     1.0      /* float */
#define c_crdh_min_range_decim_stop      1.0      /* float */
#define c_crdh_min_range_decim_step      1.0      /* float */

#define c_crdh_max_range_decim_start     512.0    /* float */
#define c_crdh_max_range_decim_stop      512.0    /* float */
#define c_crdh_max_range_decim_step      511.0    /* float */

/*-----------------------------------------------------------------------------
   fixed default and fixed range for generation options
  ----------------------------------------------------------------------------*/
/* Earth clutter power threshold         SAT HMI 7.2 */
#define c_crdh_dfl_clutter_pw_thrshd     -200.0  /* dB */
#define c_crdh_min_clutter_pw_thrshd     -250.0  /* dB */
#define c_crdh_max_clutter_pw_thrshd     -150.0  /* dB */

/* range quantization factor             SAT HMI 7.3 */
#define c_crdh_dfl_range_quant_factor    1        /* integer */
#define c_crdh_min_range_quant_factor    1        /* integer */
#define c_crdh_max_range_quant_factor    100      /* integer */

/* Earth clutter power threshold         SAT HMI 7.2 */
#define c_crdh_dfl_doppler_quant_factor 1.5       /* float */
#define c_crdh_min_doppler_quant_factor 1.0       /* float */
#define c_crdh_max_doppler_quant_factor 10.0      /* float */

```

```
/*=============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      crdh_ant.c - Antenna Model Definition I/O functions

Identification: [@(#)]

SCCS Id:        @(#)crdh_ant.c  1.1 93/05/11

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package implements read/write functions for the
                Antenna Model Definition data store.
                Note that RRA and CFA Antenna Model Definition files
                must have different file subfix in order to distinguish them.

Designer:       Stephen Tran

Change History:

   SCR  Date         Author    Change made
   ---  -----------  ------  ---------------------------------------
   n/a 26-Apr-1993   cst    Initial Release

=============================================================================*/

/* Included external declarations */
#include <stdio.h>
#include <rpc/rpc.h>
#include "spd_intf.h"           /* SPD system generic types package */
#include "crad_intf.h"          /* Satellite/Radar Definition package */
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* Implementation */
/*=============================================================================

    Function:     f_crdh_ant_read_RRA

    Description:
       This function reads a RRA Antenna Model Definition data file
       named by pathname into in-memory data structure.
       pathname points to a character string that contains the directory and
       the name of RRA Antenna Model definition file to be read.
       The memory for the data structure will be allocated by this function.
       Therefore, it is the caller's responsibility to release the memory.

    Arguments:
       pathname                 - in    - directory and file name of
                                          the RRA Antenna Model
                                          Definition file
       RRA_def                  -   out - RRA Antenna Model Definition

    External variables:
       none

    Exceptions:
       s_spd_failure returned, if the provided pathname is not found or
       can not be read.

    Notes:
     None

   High Level PDL
      BEGIN  f_crdh_ant_read_RRA
        USE fopen TO open the RRA Antenna Model Definition file
          with "read" option
        IF file does not exist or can not be read THEN
           return s_spd_failure
        ELSE
           USE xdrstdio_create TO initialize a record-oriented XDR stream
             object (xdrs) in XDR_DECODE mode, which will be used in the
             subsquent XDR calls
           USE xdr_t_crad_RRA_antenna TO read the RRA Antenna Model Definition
             file in XDR format into in-memory data structure
           IF error THEN return s_spd_failure
           USE xdr_destroy TO destroy xdrs
        ENDIF
      END f_crdh_ant_read_RRA
============================================================================*/
t_spd_status f_crdh_ant_read_RRA (
                char *pathname,
                t_crad_RRA_file *RRA_def)
{
}


/*===========================================================================

   Function:     f_crdh_ant_write_RRA

   Description:
      This function writes a RRA Antenna Model Definition data
      structure to a data file named by pathname.
      pathname points to a character string that contains the directory and
      the name of RRA Antenna Model definition file to be write.

   Arguments:
      pathname                  - in    - directory and file name of
                                          the RRA Antenna Model
                                          Definition file
      RRA_def                   - in    - Antenna Model Definition

   External variables:
      none

   Exceptions:
      s_spd_failure returned if the data can not be written to the provided
      pathname.

   Notes:
      none

   High Level PDL
      BEGIN  f_crdh_ant_write_RRA
        USE fopen TO open a RRA Antenna Model Definition file
        with "write" option
        IF file does not exist or can not be write THEN
           return s_spd_failure
        ELSE
           USE xdrstdio_create TO initialize a record-oriented XDR stream
             object (xdrs) in XDR_ENCODE, which will be used in the subsquent
             XDR calls
           USE xdr_t_crad_RRA_antenna TO write the RRA Antenna Model Definition
             structure to data file in XDR format.
           IF error THEN return s_spd_failure
           USE xdr_destroy TO destroy xdrs
        ENDIF
      END f_crdh_ant_write_RRA
============================================================================*/
t_spd_status f_crdh_ant_write_RRA (
                char *pathname,
                t_crad_RRA_file *RRA_def)
{
}
/*============================================================================

   Function:     f_crdh_ant_read_CFA

   Description:
      This function reads a CFA Antenna Model Definition data file
      named by pathname into in-memory data structure.
      pathname points to a character string that contains the directory and
      the name of CFA Antenna Model definition file to be read.
      The memory for the data structure will be allocated by this function.
      Therefore, it is the caller's responsibility to release the memory.

   Arguments:
      pathname                  - in    - directory and file name of
                                          the CFA Antenna Model
                                          Definition file
      CFA_def                   -   out - CFA Antenna Model Definition

   External variables:
      none

   Exceptions:
      s_spd_failure returned, if the provided pathname is not found or
      can not be read.

   Notes:
      None

   High Level PDL
      BEGIN  f_crdh_ant_read_CFA
        USE fopen TO open the CFA Antenna Model Definition file
          with "read" option
        IF file does not exist or can not be read THEN
           return s_spd_failure
        ELSE
           USE xdrstdio_create TO initialize a record-oriented XDR stream
             object (xdrs) in XDR_DECODE mode, which will be used in the
             subsequent XDR calls
           USE xdr_t_crad_CFA_antenna TO read the CFA Antenna Model Definition
             file in XDR format into in-memory data structure
           IF error THEN return s_spd_failure
           USE xdr_destroy TO destroy xdrs
        ENDIF
      END f_crdh_ant_read_CFA
===========================================================================*/
t_spd_status f_crdh_ant_read_CFA (
                char *pathname,
                t_crad_CFA_file *CFA_def)
{
}


/*============================================================================

   Function:     f_crdh_ant_write_CFA

   Description:
      This function writes a CFA Antenna Model Definition data
      structure to a data file named by pathname.
      pathname points to a character string that contains the directory and
      the name of CFA Antenna Model definition file to be write.

   Arguments:
      pathname                  - in    - directory and file name of
                                          the CFA Antenna Model
                                          Definition file
      CFA_def                   - in    - Antenna Model Definition

   External variables:
      none

   Exceptions:
      s_spd_failure returned if the data can not be written to the provided
      pathname.

   Notes:
      none

   High Level PDL
      BEGIN  f_crdh_ant_write_CFA
        USE fopen TO open a CFA Antenna Model Definition file
        with "write" option
        IF file does not exist or can not be write THEN
           return s_spd_failure
        ELSE
           USE xdrstdio_create TO initialize a record-oriented XDR stream
             object (xdrs) in XDR_ENCODE, which will be used in the subsquent
             XDR calls
           USE xdr_t_crad_CFA_antenna TO write the CFA Antenna Model Definition
             structure to data file in XDR format.
           IF error THEN return s_spd_failure
           USE xdr_destroy TO destroy xdrs
        ENDIF
      END f_crdh_ant_write_CFA
===========================================================================*/
t_spd_status f_crdh_ant_write_CFA (
                char *pathname,
                t_crad_CFA_file *CFA_def)
{
}
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      crdh_initialize.c - Satellite/Radar Definition HMI task
                                    initialization functions

Identification: [@(#)]

SCCS Id:        @(#)crdh_initialize.c   1.3 93/09/18

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package implements initialization functions for the
                Satellite/Radar Definition Generation task.  These functions
                initialize a data structure of a Satellite/Radar Definiton,
                Antenna Model (RRA and CFA), and subintervals (dwell interval,
                NCI or CPI interval).

Designer:       Stephen Tran

Change History:

   SCR  Date        Author   Change made
   ---  ----------  ------   ----------------------------------------
   n/a 26-Apr-1993  cst     Initial Release

=======================================================================*/

/* Included external declarations */
#include <stdio.h>
#include <rpc/rpc.h>
#include "spd_intf.h"           /* SPD system generic types package */
#include "crad_intf.h"          /* Satellite/Radar Definition package */
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* Implementation */
/*========================================================================

   Function:    f_crdh_init_RRA

   Description:
      This function initializes a RRA Antenna Model Definition data
      structure.  It provides valid defaults for all attributes of
      a RRA antenna data structure including physical and operational
      characteristics.

   Arguments:
      RRA_def                  -   out - RRA Antenna Model Definition

   External variables:
      none

   Exceptions:
      None

   Notes:
      None

   High Level PDL
      BEGIN  f_crdh_init_RRA
```

```
         assign defaults to all parameters of the RRA antenna data structure
      END f_crdh_init_RRA
==========================================================================*/
void f_crdh_init_RRA (
              t_crad_antenna_model *RRA_model)
{
}

/*==========================================================================

   Function:    f_crdh_init_CFA

   Description:
      This function initializes a CFA Antenna Model Definition data
      structure.  It provides valid defaults for all attributes of
      a CFA antenna data structure including physical and operational
      characteristics.

   Arguments:
      CFA_def                     -   out - CFA Antenna Model Definition

   External variables:
      none

   Exceptions:
      None

   Notes:
      None

   High Level PDL
      BEGIN  f_crdh_init_CFA
         assign defaults to all parameters of the CFA antenna data structure
      END f_crdh_init_CFA
==========================================================================*/
void f_crdh_init_CFA (
              t_crad_antenna_model *CFA_def)
{
}

/*==========================================================================

   Function:    f_crdh_init_CPI

   Description:
      This function initializes a set of Coherent Processing Interval
      (CPI) by providing default parameter values to all intervals.
      This function allocates memory to store the structure array of CPIs;
      the number of the CPI interval entries is returned.
      For RRA, number of CPIs is 4 and CFA is 6.

   Arguments:
      concept                     - in    - concept (RRA or CFA)
      CPI_len                     -   out - number of CPI intervals
      CPI_val                     -   out - an array of CPI interval elements
                                            with CPI_len entries

   External variables:
      none

   Exceptions:
      None

   Notes:
      None

   High Level PDL
      BEGIN  f_crdh_init_CPI
         CPI_len = default number of CPIs
         FOR all CPIs DO
```

```
            assign defaults to all parameters of one CPI
         ENDFOR
      END f_crdh_init_CPI
======================================================================*/
void f_crdh_init_CPI (
               t_spd_concept   concept,
               t_spd_int32     CPI_len,
               t_crad_CPI_elm  *CPI_val)
{
}
/*======================================================================

   Function:    f_crdh_init_NCI

   Description:
      This function initializes a set of Non-Coherent Integration
      (NCI) by providing default parameter values to all intervals.
      This function allocates memory to store the structure array of NCIs;
      the number of the NCI interval entries is returned.
      Currently, the default number of NCIs is 3 for both RRA and CFA.

   Arguments:
      NCI_len                 -   out - number of NCI intervals
      NCI_val                 -   out - an array of NCI interval elements
                                        with NCI_len entries

   External variables:
      none

   Exceptions:
      None

   Notes:
      None

   High Level PDL
      BEGIN  f_crdh_init_NCI
         NCI_len = default number of NCIs
         FOR all NCIs DO
            assign defaults to all parameters of one NCI
         ENDFOR
      END f_crdh_init_NCI
======================================================================*/
void f_crdh_init_NCI (
               t_spd_int32     NCI_len,
               t_crad_NCI_elm  *NCI_val)
{
}

/*======================================================================

   Function:    f_crdh_init_DI

   Description:
      This function initializes a set of Dwell Interval (DI) by providing
      default parameter values to all intervals.
      This function allocates memory to store the structure array of dwells;
      the number of the dwell interval entries is returned.
      Currently, the default number of DIs is 1 for both RRA and CFA.

   Arguments:
      CPI_len                 -   out - number of CPI intervals
      CPI_val                 -   out - an array of CPI interval elements
                                        with CPI_len entries

   External variables:
      none

   Exceptions:
      None

   Notes:
      None

   High Level PDL
      BEGIN  f_crdh_init_DI
         CPI_len = 1 (default number of dwells)
         FOR all dwells DO
            assign defaults to all parameters of one DI
         ENDFOR
      END f_crdh_init_DI
===========================================================================*/
void f_crdh_init_DI (
                t_spd_int32     DI_len,
                t_crad_DI_elm   *DI_val)
{
}
```

```
/*===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      crdh_main.c - Main program of Controller Satellite/Radar
                              Definition HMI task.

Identification: [@(#)]

SCCS Id:        @(#)crdh_main.c 1.1 93/04/22

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file implements the main program for the
                Satellite/Radar Definition HMI task.


Designer:       Stephen Tran

Change History:

   SCR  Date        Author   Change made
   ---  ----------- ------  -----------------------------------------
   n/a  29-Jan-1993  cst    Initial Release

===========================================================================*/

/* Included external declarations */
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include "spd_intf.h"
#include "crad_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* common constants */
/* none */

/* Implementation */
/*===========================================================================

   program:     crdh_main

   Description:
      Main program to run Controller Satellite/Radar Definition HMI task.

      This is a stand-alone task which provides capability to define
      a satellite/radar definition through Motif widgets and dialogues
      in X window environment.

      This task is implemented using UIM/X tool with X and Motif widgets.
      UIM/X is a user interface management tool; all code can be entered
      without leaving the application itself and UIM/X provides a text
      editor function for editing C code. All widget resources (core,
      specific, behavior, declaration, constraint) defined for an interface
      are kept in an interface file (.i); one file per interface.

      These interface files are created automatically by UIM/X. The C code
      is created either calling uxcgen utility from the UNIX command line
```

DRAFT

```
        or pressing the "Write C code" button in UIM/X. Typically, the
        interface files are submitted for Configuration Management and
        the C code is created for compiling and linking via makefile.

        In essence, most of the CRDH CSU codes are implemented in X callback
        functions (how the widget reacts to events). The UIM/X tool supports
        the generation of the callback function as well as the widget
        creation and display function in C code; one .C file per interface.
        We have to compile the generated C code and link them into application
        executable.

        This task will have a main program, which calls various initialization
        routines and then enters XtAppMainLoop, awaiting x events and never
        to return.  The user can terminate this task by selecting the "EXIT"
        menu.

        A Satellite/Radar top-level visualization window (a control pannel
        window) will be displayed. This visualization window shows the
        Satellite/Radar Definition structure and provides capability to
        define a definition by applying the edit operation to parameter groups.

        If the user selects "OPEN" menu, a file list is displayed for selection.
        After an existing file is selected, the definition file is read into
        in-memory definition structure.  On the other hand, if the user selects
        "NEW" menu, the in-memory structure will be initialized with default
        parameter values.  For example, by defaults a definition will have
        only one dwell, three NCIs, each NCI will have 4 CPIs (RRA) or
        6 CPIs (CFA).  The definition structure is defined in the package CRAD.

        Other low-level dialog windows will be displayed as a result of
        some widget selection in the top-level visualization window.
        Each parameter definition will be implemented in a form of a MOTIF
        widget (push button, option button, scaler, text, etc.)
        A callback function is invoked every time a parameter value is
        modified.  Note that defaults are always provided for a new definition
        or for a new interval (DI, NCI, CPI).

        An instance of interval can be selected and then applied to an editing
        function (create, delete, copy or edit).  For example, the user can
        create a new dwell, delete a dwell, copy dwell #1 to #2, or edit
        a dwell interval.  The instances of intervals are stored, locally,
        as linked list for easy of manipulation. The linked list is converted
        into the in-memory definition data structure for output or
        vice versa for input.

        Callback functions extract parameter values from a in-memory structure
        for display as well as extract parameter values from the display
        and place them back into the in-memory definition structure when
        the user hits the "OK" button in the dialog window. No update is
        performed in the in-memory structure, if the user presses the "CANCEL"
        button.

        Callback functions also perform range validation, implements
        structural dependency logic as well as some parameter dependency
        logic by ghosting the widgets not associated with some previously
        entered data. See Generation HMI table.

        Most of range validation and parameter dependency checking
        will be performed using functions provided by package CVAL.
        Validation will be invoked, every time when the user presses
        the "VALIDATE" menu button. The validation message log will
        be displayed and the user can optionally save the validation log
        to a user-specified file.

        After the definition is completed, the user can select "SAVE" or
        "SAVE-AS" menu to write out the in-memory defintion structure
        into a UNIX external data file in XDR format.  Note that
        the definition structure is defined in the CRAD package and
        the XDR routines for external data convertion are provided by
        the same package.
```

```

      Note that this task also provides the function to save or read
      a re-used Antenna Model Definition to/from external file.

  Command:

         crdh_main 2>log_pathname &

      This task will log unsolicited errors occured in X/Motif calls
      to standard error and the error messages will then be
      re-directed to the user-specified pathname.

   Global variables:
      sat_def          - in-memory Satellite/Radar Definition data structure

   Exceptions:
      none

   Notes:
      none

   High Level PDL

      BEGIN crdh_main
         USE f_CERR_initialize() TO initialize log file
         get node names of Controller and RSS from environment variables:
           CTLR_NODE and RSS_NODE
         get default directory name for Satellite/Radar Definition file
         USE XtAppInitialize TO initialize the X Application
         USE UIM/X routines TO display the top-level visualization window
         USE XtAppMainLoop() TO enter infinite loop and dispatch X event
           to the appropriate callback functions
      END crdh_main

      BEGIN user interface services functions
         { these functions will be implemented in different callback functions }

         case user presses "OPEN" :
            user selects Satellite/Radar Definition pathname (directory +
              filename) through a selection list
            USE f_crad_read TO read the specified data file into
              in-memory definition data structure

         case user presses "NEW" :
            initialize the in-memory definition data structure

         case user presses "SAVE" :
            USE f_crad_write TO write out the in-memory data structure to
              external data file

         case user presses "SAVE-AS" :
            user enters directory name and file name for the Satellite/Radar
              Definition data file
            USE f_crad_write TO write out the in-memory data structure to
              external data file

         case user presses "VALIDATE" :
            USE f_cval_sat_rad_def TO validate the in-memory data structure
            IF error THEN
               display the validation log to the error message area
            ENDIF

      END user interface service functions
    End of PDL
==========================================================================*/

/* Routine Declaration */
t_spd_status main (
      t_spd_int32        argc,
      char               **argv,
        char                **envp)
{   /* Begin of main */
}   /* End of main */
```

THIS PAGE INTENTIONALLY LEFT BLANK

D1.2 Environment HMI (CENH)

THIS PAGE INTENTIONALLY LEFT BLANK

```
[][][][][][][][][][][][][][][][][][][][][][][][][][][][]
[][][][]                                        [][][][]
[][][][]                                        [][][][]
[][][][]                                        [][][][]
  [][]                                            [][]
  [][]              CDADL PROCESSOR               [][]
  [][]                                            [][]
  [][]                                            [][]
  [][]               VERSION 2.4.6                [][]
  [][]                                            [][]
  [][]          SERIAL NUMBER: C38U0011           [][]
  [][]                                            [][]
  [][]              AUTHORIZED USER:              [][]
  [][]             Raytheon - Canada              [][]
  [][]                                            [][]
  [][]                                            [][]
  [][]                                            [][]
  [][]                                            [][]
  [][]                                            [][]
[][][][]                                        [][][][]
[][][][][][][][][][][][][][][][][][][][][][][][][][][][]
```

```
 1      FILE cenh_close.c
 2      FILE cenh_edit_object.c
 5      FILE cenh_handle_lmouse.c
 9      FILE cenh_handle_mmouse.c
12      FILE cenh_handle_rmouse.c
14      FILE cenh_init.c
16      FILE cenh_empty_fov.c
17      FILE cenh_load.c
19      FILE cenh_ld_radar.c
20      FILE cenh_main.c
34      FILE cenh_redraw.c
36      FILE cenh_save.c
37      FILE cenh_select.c
38      FILE cenh_valid.c
43      REPORT TBD CROSS REFERENCE REPORT
44      REPORT INVOCATION TREE
45      REPORT ERROR SUMMARY
```

```
/*%%===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_close.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR   Date        Author   Change made
---  -----------  ------  ---------------------------------------------


CSU_NAME:     HMI

FUNCTION:     f_cenh_close

PURPOSE:    This function is used to close down the data entry screens
and PHIGS f_o_r Environment Definition parameter entry task.



INPUT                             SOURCE
-----                             ------

OUTPUT                             DESTINATION
------                             -----------


CREATION DATE:     June 03 1993

==============================================================================*/
#include "cenh.h"

void f_cenh_close(void)
    /* Environment Definition HMI close function */
{
/*|

    USE dialog_box to ask user if they are "sure they wish to quit"

    IF (dialog_response==TRUE) THEN
    |
    |   IF (data_status==CHANGED) THEN
    |   |   popup question dialog "Do you wish to save first?"
    |   |   IF (query indicates SAVE) THEN
    |   |   |   display a file selection dialog
    |   |   |   USE f_cenh_save (&file_name) TO write database into file
    |   |   ENDIF
    |   ENDIF
    |
    |   USE pclose_ws(WS_ID) TO close the workstation
    |
    |   USE pclose_phigs() TO close PHIGS
    |
    |   USE UxDestroyWidget() TO destroy the top level screen widget
    |
    |   USE exit(0) to quit the application
    |
    ENDIF
*/
}

/* end of cenh_close.c */
```

```
/*%%========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_edit_object.c

Identification: [@(#)]

SCCS Id:        %M% %I%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR   Date         Author   Change made
---   -----------  ------   ---------------------------------------------------


CSU_NAME:      HMI

FUNCTION:      f_cenh_edit_object

PURPOSE:    This function will modify the PHIGS CSS data structure which
is a display list of what is displayed in the FOV display.


INPUT                                       SOURCE
-----                                       ------

OUTPUT                                       DESTINATION
------                                       -----------


CREATION DATE:      June 03 1993

==========================================================================*/
#include "cenh.h"

void f_cenh_edit_object(int data_type, t_cend_object object, t_spd_int32 object_id, int edit_mode)
    /* Environment Definition HMI close function */
{
/*|
    Ppoint *object_vert;
    Ppoint_list object_points;
    Pdata object_data;

    USE popen_struct() TO open the FOV structure

    USE pset_elem_ptr_label(data_type) TO set the pointer to the requested object definitions

    SWITCH edit_mode
    |
    |   CASE c_add_element:
    |   |
    |   |   USE pset_edit_mode(PEDIT_INSERT) TO set force insert mode
    |   |   object_data.data = (void *)&object_id
    |   |   object_data.size = sizeof (object_id)
    |   |   USE pappl_data(&object_id) TO store the ID number with the point array for
    |   |   later editing
    |   |
    |   |   SWITCH data_type
    |   |   |
    |   |   |   CASE c_discrete:
    |   |   |   |
    |   |   |   |   USE object_points.points=malloc() to allocate enough space f_o_r vertex definition
    |   |   |   |
    |   |   |   |   USE poffset_elem_ptr()  to move past colour specs. etc.
    |   |   |   |
    |   |   |   |   object_points.num_points = object.discrete->vertex_count;
    |   |   |   |
    |   |   |   |   convert verticies from lat/long to world coords on 2-D plane
    |   |   |   |   FOR (vertex_num=0 TO vertec_num<object.discrete->vertex_count) LOOP
    |   |   |   |   |
    |   |   |   |   |   USE cenh_convert_to_xy(&object_points.points[vertex_num],
    |   |   |   |   |   &object.discrete ->loc[vertex_num]) TO convert Lat/Long coord into x-y in meters
    |   |   |   |   |
|   |   |   |   ENDLOOP
|   |   |   |
|   |   |   |   USE pfill_area (object_points) TO add the filled polygon to the list
|   |   |   |
|   |   |   |   USE free(object_points) TO return memory to free pool
|   |   |   |
|   |   |   CASE c_patch:
|   |   |   |   USE poffset_elem_ptr()  to move past colour specs. etc.
|   |   |   |
|   |   |   |   patch_points.num_points = object.patch->vertex_count;
|   |   |   |
|   |   |   |   convert verticies from lat/long to world coords on 2-D plane
|   |   |   |
|   |   |   |   FOR (vertex_num=0 TO vertec_num<object.patch->vertex_count) LOOP
|   |   |   |   |
|   |   |   |   |   USE cenh_convert_to_xy(&patch_points.points[vertex_num], &patch->loc[vertex_num])
|   |   |   |   |   TO convert Lat/Long coord into x-y in meters
|   |   |   |   |
|   |   |   |   ENDLOOP
|   |   |   |
|   |   |   |   USE pfill_area (patch_points) TO add the filled polygon to the list
|   |   |   |
|   |   |   |   USE free(patch_points) TO return memory to free pool
|   |   |   |
|   |   |   CASE c_chaff:
|   |   |   |   USE object_points.points=malloc() to allocate enough space f_o_r marker definition
|   |   |   |
|   |   |   |   USE poffset_elem_ptr()  to move past colour specs. etc.
|   |   |   |   note that pset_marker_type() was used to set the marker at init
|   |   |   |
|   |   |   |   object_points.num_points = 1
|   |   |   |
|   |   |   |   convert lat/long to world coords on 2-D plane
|   |   |   |
|   |   |   |   USE cenh_convert_to_xy(object_points.points, &object.chaff ->loc) TO
|   |   |   |   convert Lat/Long coord into x-y in meters
|   |   |   |
|   |   |   |   USE ppolymarker (&object_points) TO add polymarker to the list
|   |   |   |
|   |   |   |   USE free(object_points) TO return memory to free pool
|   |   |   |
|   |   |   |
|   |   |   CASE c_cell:
|   |   |   |   USE object_points.points=malloc() to allocate enough space f_o_r marker definition
|   |   |   |
|   |   |   |   USE poffset_elem_ptr()  to move past colour specs. etc.
|   |   |   |   note that pset_marker_type() was used to set the marker at init
|   |   |   |
|   |   |   |   object_points.num_points = 1
|   |   |   |
|   |   |   |   convert lat/long to world coords on 2-D plane
|   |   |   |
|   |   |   |   USE cenh_convert_to_xy(object_points.points, &object.cell->loc) TO
|   |   |   |   convert Lat/Long coord into x-y in meters
|   |   |   |
|   |   |   |   USE pset_marker_size() to scale the polymarker depending on
|   |   |   |   object.cell->radius
|   |   |   |
|   |   |   |   USE ppolymarker (&object_points) TO add polymarker to the list
|   |   |   |
|   |   |   |   USE free(object_points) TO return memory to free pool
|   |   |   |
|   |   |   CASE c_jammer:
|   |   |   |
|   |   |   |   USE object_points.points=malloc() to allocate enough space f_o_r marker definition
|   |   |   |
|   |   |   |   USE poffset_elem_ptr()  to move past colour specs. etc.
|   |   |   |   note that pset_marker_type() was used to set the marker at init
|   |   |   |
|   |   |   |   object_points.num_points = 1
|   |   |   |
|   |   |   |   convert lat/long to world coords on 2-D plane
|   |   |   |
|   |   |   |   USE cenh_convert_to_xy(object_points.points, &object.jammer->loc) TO
|   |   |   |   convert Lat/Long coord into x-y in meters
|   |   |   |
|   |   |   |   USE ppolymarker (&object_points) TO add polymarker to the list
|   |   |   |
|   |   |   |   USE free(object_points) TO return memory to free pool
|   |   |   |
|   |   |   CASE c_target:
|   |   |   |
|   |   |   |   USE object_points.points=malloc() to allocate enough space f_o_r marker definition
```

```
    |   |   |   |
    |   |   |   |   USE poffset_elem_ptr()  to move past colour specs. etc.
    |   |   |   |   note that pset_marker_type() was used to set the marker at init
    |   |   |   |
    |   |   |   |   object_points.num_points = 1
    |   |   |   |
    |   |   |   |   convert lat/long to world coords on 2-D plane
    |   |   |   |
    |   |   |   |   USE canh_convert_to_xy(object_points.points, &object.target->loc) TO
    |   |   |   |   convert Lat/Long coord into x-y in meters
    |   |   |   |
    |   |   |   |   USE ppolymarker (&object_points) TO add polymarker to the list
    |   |   |   |
    |   |   |   |   USE free(object_points) TO return memory to free pool
    |   |   |   |
    |   |   |   CASE c_current: this is a temporary object used f_o_r editing current object by drawing
    |   |   |   |   symbols at vertex locations
    |   |   |   |
    |   |   |   |
    |   |   ENDSWITCH
    |   |
    |   CASE c_delete_element:
    |   |
    |   |   search forward through the data to find a matching clutter object
    |   |   WHILE (matching element not found) LOOP
    |   |   |
    |   |   |   USE pinq_elem_content() TO retrieve object_id data f_o_r stored clutter object
    |   |   |   IF (object_id matches stored object ID) THEN
    |   |   |   |   match found
    |   |   |   ELSE
    |   |   |   |   there was an error searching f_o_r the object
<-------------------return
    |   |   |   ENDIF
    |   |   ENDLOOP
    |   |   USE poffset_elem_ptr(1) TO move the element pointer down one
    |   |   USE pdel_elem() to delete the current element
    |   |
    |   CASE c_replace_element:
    |   |
    |   |   USE f_canh_edit_object(data_type, object, object_id, c_delete_element) TO delete the
    |   |   object
    |   |   USE f_canh_edit_object(data_type, object, object_id, c_add_element) TO add the
    |   |   object
    |   |
    ENDSWITCH
*/
}

/* end of canh_edit_object.c */
```

```
/*%%=======================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_handle_lmouse.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR   Date         Author   Change made
---   ----------   ------   -----------------------------------------------


CSU_NAME:      HMI

FUNCTION:      f_cenh_handle_lmouse

PURPOSE:    This is the handler function f_o_r the left mouse button.  The
action is dependant on the current state of the FOV editor.

                                    SOURCE
INPUT                               ------
-----
                                     DESTINATION
OUTPUT                               -----------
------


CREATION DATE:      June 03 1993
=========================================================================*/
#include "cenh.h"

void f_cenh_handle_lmouse(void)
    /* Environment Definition HMI handle_lmouse function */
{
/*|

    get the cursor position in WC at the time the button was pressed

    determine which type of data is currently being edited
    SWITCH active_type
    |
    | . CASE c_clutter:
    |   |
    |   |   set data_status=CHANGED
    |   |   determine what the current edit mode is
    |   |   SWITCH edit_mode
    |   |   |
    |   |   |   CASE c_add: add verticies to the current polygon
    |   |   |   |   IF (this is at least the third point added) THEN
    |   |   |   |   |   determine i_f the new vertex location is legal f_o_r a simple polygon
    |   |   |   |   |   IF (polygon is legal) THEN
    |   |   |   |   |   |   save the new vertex location in a temporary t_cend_clutter_patch struct
    |   |   |   |   |   |   USE f_cenh_edit_object(c_current,&t_cend_clutter_patch,patch_count,
    |   |   |   |   |   |   c_add_element) TO add a clutter patch vertex to the CSS
    |   |   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
    |   |   |   |   |   ELSEIF (vertex_num<c_cend_max_vertices) THEN
    |   |   |   |   |   |   popup a warning dialog box indicating "New vertex outside of range
    |   |   |   |   |   |   f_o_r a simple polygon... please re-enter"
    |   |   |   |   |   ELSE
    |   |   |   |   |   |   popup a warning dialog box indicating "Cannot enter more than the
    |   |   |   |   |   |   maximum of TBD vertecies"
    |   |   |   |   |   ENDIF
    |   |   |   |   ELSE
    |   |   |   |   |   save the new vertex location in a temporary t_cend_clutter_patch struct
    |   |   |   |   |   USE f_cenh_edit_object(c_current,&t_cend_clutter_patch,patch_count,
    |   |   |   |   |   c_add_element) TO add a clutter patch vertex to the CSS
    |   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
    |   |   |   |   ENDIF
    |   |   |   |
|   |   |   CASE c_delete: delete the entire polygon associated with the closest vertex
|   |   |   |   remove the clutter patch definition from the database
|   |   |   |   USE f_cenh_edit_object(c_clutter,&t_cend_clutter_patch,patch_count,
|   |   |   |   c_delete_element) TO remove a clutter patch from the CSS
|   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |
|   |   |   CASE c_delete_vertex: delete the closest vertex on the current polygon
|   |   |   |   remove the vertex from the database
|   |   |   |   USE f_cenh_edit_object(c_patch,&t_cend_env_def.patch[patch_count],patch_count,
|   |   |   |   c_replace_element) TO redraw the new clutter patch in the CSS
|   |   |   |   set edit_mode=c_no_op
|   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |
|   |   |   CASE c_new: begin the declaration of a new polygon
|   |   |   |   save the new vertex location in a temporary t_cend_clutter_patch struct
|   |   |   |   add this to the selection list using the the first vertex as lat/long
|   |   |   |   USE f_cenh_edit_object(c_current,&t_cend_clutter_patch,patch_count,
|   |   |   |   c_add_element) TO add a clutter patch vertex to the CSS
|   |   |   |
|   |   |   CASE c_modify: select the edge in which to insert a new vertex
|   |   |   |   set edit_mode = c_modify_point
|   |   |   |   store the co-ords of the vertices between which the new vertex will be located
|   |   |   |
|   |   |   CASE c_modify_point: specify the co-ords of the new point
|   |   |   |   determine i_f the new vertex location is legal f_o_r a simple polygon
|   |   |   |   IF (polygon is legal) THEN
|   |   |   |   |   insert the new point into the vertex array of the clutter definition
|   |   |   |   |   USE f_cenh_edit_object(c_patch,&t_cend_env_def.patch[patch_count],patch_count,
|   |   |   |   |   c_replace_element) TO add a clutter patch to the CSS
|   |   |   |   |   set edit_mode=c_no_op
|   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |   ELSE
|   |   |   |   |   popup a warning dialog box indicating "New vertex outside of range
|   |   |   |   |   f_o_r a simple polygon... please re-enter"
|   |   |   |   ENDIF
|   |   |   |
|   |   |   CASE c_copy:
|   |   |   |   make a new instance of a clutter patch
|   |   |   |   force user to enter a patch-ID
|   |   |   |
|   |   |   CASE c_select:
|   |   |   |   cause the patch associated with the closest vertex to become the current object
|   |   |   |   USE f_cenh_select_list() TO highlight the selection list
|   |   |   |
|   |   ENDSWITCH
|   |
|   CASE c_discrete:
|   |   set data_status=CHANGED
|   |
|   |   determine what the current edit mode is
|   |   SWITCH edit_mode
|   |   |
|   |   |   CASE c_delete: delete the closest discrete clutter data
|   |   |   |   remove the discrete clutter from the database
|   |   |   |   USE f_cenh_edit_object(c_discrete,&t_cend_env_def.discrete[discrete_count],
|   |   |   |   discrete_count, c_delete_element) TO remove the discrete clutter from the CSS
|   |   |   |   update the Env_Discrete_Clutter_Iface to indicate data deleted
|   |   |   |   set edit_mode=c_no_op
|   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |
|   |   |   CASE c_select:
|   |   |   |   cause the closest discrete clutter to become the current object
|   |   |   |   USE f_cenh_select_list() TO highlight the selection list
|   |   |   |
|   |   |   |
|   |   ENDSWITCH
|   |
|   CASE c_chaff:
|   |   set data_status=CHANGED
|   |
|   |   determine what the current edit mode is
|   |   SWITCH edit_mode DO
|   |   |
|   |   |   CASE c_delete: delete the closest chaff bundle data
|   |   |   |   remove the chaff bundle from the database
|   |   |   |   USE f_cenh_edit_object(c_chaff,&t_cend_env_def.chaff[chaff_count],
|   |   |   |   chaff_count, c_delete_element) TO remove the chaff bundle from the CSS
|   |   |   |   update the Env_Chaff_Iface to indicate data deleted
|   |   |   |   set edit_mode=c_no_op
|   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |
|   |   |   CASE c_select:
|   |   |   |   cause the closest chaff bundle to become the current object
```

```
    |   |   |   |   USE f_cenh_select_list() TO highlight the selection list
    |   |   |   |
    |   |   ENDSWITCH
    |   |
    |   CASE c_cell:
    |   |   set data_status=CHANGED
    |   |
    |   |   determine what the current edit mode is
    |   |   SWITCH edit_mode DO
    |   |   |
    |   |   |   CASE c_delete: delete the closest precipitation cell data
    |   |   |   |   remove the precipitation cell from the database
    |   |   |   |   USE f_cenh_edit_object(c_cell,&t_cend_env_def.cell[cell_count],
    |   |   |   |   cell_count, c_delete_element) TO remove the precipitation cell from the CSS
    |   |   |   |   update the Env_Cell_Iface to indicate data deleted
    |   |   |   |   set edit_mode=c_no_op
    |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
    |   |   |   |
    |   |   |   CASE c_select:
    |   |   |   |   cause the closest precipitation cell to become the current object
    |   |   |   |   USE f_cenh_select_list() TO highlight the selection list
    |   |   |   |
    |   |   ENDSWITCH
    |   |
    |   |
    |   CASE c_jammer:
    |   |
    |   |   set data_status=CHANGED
    |   |
    |   |   determine what the current edit mode is
    |   |   SWITCH edit_mode DO
    |   |   |
    |   |   |   CASE c_delete: delete the closest jammer data
    |   |   |   |   remove the jammer from the database
    |   |   |   |   USE f_cenh_edit_object(c_jammer,&t_cend_env_def.jammer[jammer_count],
    |   |   |   |   jammer_count, c_delete_element) TO remove the jammer from the CSS
    |   |   |   |   update the Env_Jammer_Iface to indicate data deleted
    |   |   |   |   set edit_mode=c_no_op
    |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
    |   |   |   |
    |   |   |   CASE c_select:
    |   |   |   |   cause the closest jammer to become the current object
    |   |   |   |   USE f_cenh_select_list() TO highlight the selection list
    |   |   |   |
    |   |   ENDSWITCH
    |   |
    |   CASE c_target:
    |   |
    |   |   set data_status=CHANGED
    |   |
    |   |   determine what the current edit mode is
    |   |   SWITCH edit_mode DO
    |   |   |
    |   |   |   CASE c_delete: delete the closest target data
    |   |   |   |   remove the target from the database
    |   |   |   |   USE f_cenh_edit_object(c_target,&t_cend_env_def.target[target_count],
    |   |   |   |   target_count, c_delete_element) TO remove the target from the CSS
    |   |   |   |   update the Env_Target_Iface to indicate data deleted
    |   |   |   |   set edit_mode=c_no_op
    |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
    |   |   |   |
    |   |   |   CASE c_select:
    |   |   |   |   cause the closest target to become the current object
    |   |   |   |   USE f_cenh_select_list() TO highlight the selection list
    |   |   |   |
    |   |   ENDSWITCH
    |   |
    |   CASE c_zoom:
    |   |
    |   |   determine what the current edit mode is
    |   |   SWITCH edit_mode DO
    |   |   |
    |   |   |   CASE c_new: begin the declaration of a zoom region
    |   |   |   |
    |   |   |   |   save the new vertex location in a temporary t_cend_zoom struct
    |   |   |   |   USE f_cenh_edit_object(c_current,&t_cend_zoom,zoom_count,
    |   |   |   |   c_add_element) TO add first zoom vertex to the CSS
    |   |   |   |   set edit_mode=c_add
    |   |   |   |
    |   |   ENDSWITCH
    |   |
    ENDSWITCH
*/
```

```
}

/* end of cenh_handle_lmouse.c */
```

```
/*%%=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_handle_mmouse.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR   Date        Author   Change made
---  ----------- ------  ---------------------------------------------


CSU_NAME:      HMI

FUNCTION:      f_cenh_handle_mmouse

PURPOSE:    This is the handler function f_o_r the middle mouse button.  The
action is dependant on the current state of the FOV editor.


INPUT                                 SOURCE
-----                                 ------

OUTPUT                                 DESTINATION
------                                 -----------


CREATION DATE:     June 03 1993

===========================================================================*/
#include "cenh.h"

void f_cenh_handle_mmouse(void)
     /* Environment Definition HMI handle_mmouse function */
{
/*|

     IF (cursor_frozen=TRUE) THEN
     |   set cursor_frozen = FALSE
     |   resume standard cursor operation within FOV
     |   set current cursor position to frozen position
 <-------return
     ENDIF

     get the current cursor position in WC

     determine which type of data is currently being edited
     SWITCH active_type
     |
     |   CASE c_clutter:
     |   |
     |   |   set data_status=CHANGED
     |   |
     |   |   determine what the current edit mode is
     |   |   SWITCH edit_mode
     |   |   |
     |   |   |   CASE c_add: the last entered point completes the polygon
     |   |   |   |   IF (at least the three points have been added) THEN
     |   |   |   |   |   USE f_cenh_edit_object(c_patch,&t_cend_env_def.patch[patch_count],patch_count,
     |   |   |   |   |   c_add_element) TO add a clutter patch to the CSS
     |   |   |   |   |   USE f_cenh_edit_object(c_current,&t_cend_clutter_patch,patch_count,
     |   |   |   |   |   c_delete_element) TO delete the temporary patch from the CSS
     |   |   |   |   |   USE f_cenh_redraw() TO redraw the FCV display with the new data
     |   |   |   |   ELSE
     |   |   |   |   |   popup warning dialog indicating "At least 3 vertices must be specified"
     |   |   |   |   ENDIF
     |   |   |   |
     |   |   |   CASE default:
     |   |   |   |   de-select the current item
     |   |   |   |   clear the data_entry popup
|    |   |   |
|    |   ENDSWITCH
|    |
|    CASE c_discrete:
|    |
|    |   set data_status=CHANGED
|    |
|    |   determine what the current edit mode is
|    |   SWITCH edit_mode
|    |   |
|    |   |   CASE c_new: declare a new discrete clutter location
|    |   |   |   save the new location in the database
|    |   |   |   USE f_cenh_edit_object(c_discrete, t_cend_env_def.discrete[discrete_count],
|    |   |   |   discrete_count,c_add_element) TO add discrete clutter to the CSS
|    |   |   |   update the data entry popup
|    |   |   |
|    |   |   CASE default:
|    |   |   |   de-select the current item
|    |   |   |   clear the data_entry popup
|    |   |   |
|    |   ENDSWITCH
|    |
|    CASE c_chaff:
|    |
|    |   set data_status=CHANGED
|    |
|    |   determine what the current edit mode is
|    |   SWITCH edit_mode
|    |   |
|    |   |   CASE c_new: declare a new chaff bundle location
|    |   |   |   save the new location in the database
|    |   |   |   USE f_cenh_edit_object(c_chaff, t_cend_env_def.chaff[chaff_count],
|    |   |   |   chaff_count,c_add_element) TO add chaff bundle to the CSS
|    |   |   |   update the data entry popup
|    |   |   |
|    |   |   CASE default:
|    |   |   |   de-select the current item
|    |   |   |   clear the data_entry popup
|    |   |   |
|    |   ENDSWITCH
|    |
|    CASE c_cell:
|    |
|    |   set data_status=CHANGED
|    |
|    |   determine what the current edit mode is
|    |   SWITCH edit_mode
|    |   |
|    |   |   CASE c_new: declare a new precipitation cell location
|    |   |   |   save the new location in the database
|    |   |   |   USE f_cenh_edit_object(c_cell, t_cend_env_def.cell[cell_count],
|    |   |   |   cell_count,c_add_element) TO add precipitation cell to the CSS
|    |   |   |   update the data entry popup
|    |   |   |
|    |   |   CASE default:
|    |   |   |   de-select the current item
|    |   |   |   clear the data_entry popup
|    |   |   |
|    |   ENDSWITCH
|    |
|    CASE c_jammer:
|    |
|    |   set data_status=CHANGED
|    |
|    |   determine what the current edit mode is
|    |   SWITCH edit_mode
|    |   |
|    |   |   CASE c_new: declare a new jammer location
|    |   |   |   save the new location in the database
|    |   |   |   USE f_cenh_edit_object(c_jammer, t_cend_env_def.jammer[jammer_count],
|    |   |   |   jammer_count,c_add_element) TO add jammer to the CSS
|    |   |   |   update the data entry popup
|    |   |   |
|    |   |   CASE default:
|    |   |   |   de-select the current item
|    |   |   |   clear the data_entry popup
|    |   |   |
|    |   ENDSWITCH
|    |
|    CASE c_target:
|    |
|    |   set data_status=CHANGED
|    |
    |   |   determine what the current edit mode is
    |   |   SWITCH edit_mode
    |   |   |
    |   |   |   CASE c_new: declare a new target location
    |   |   |   |   save the new location in the database
    |   |   |   |   USE f_cenh_edit_object(c_target, t_cend_env_def.target[target_count],
    |   |   |   |   target_count,c_add_element) TO add target to the CSS
    |   |   |   |   update the data entry popup
    |   |   |   |
    |   |   |   CASE default:
    |   |   |   |   de-select the current item
    |   |   |   |   clear the data_entry popup
    |   |   |   |
    |   |   ENDSWITCH
    |   |
    |   CASE c_zoom:
    |   |
    |   |   determine what the current edit mode is
    |   |   SWITCH edit_mode
    |   |   |
    |   |   |   CASE c_add: opposite corner to be used f_o_r zoom region
    |   |   |   |
    |   |   |   |   USE f_cenh_edit_object(c_current,&t_cend_zoom,zoom_count,
    |   |   |   |   c_delete_element) TO remove the zoom corner indicator
    |   |   |   |   determine a best-fit square and update the viewing transform to new values
    |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
    |   |   |   |
    |   |   ENDSWITCH
    |   |
    ENDSWITCH
*/
}

/* end of cenh_handle_mmouse.c */
```

```
/*%%==========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_handle_rmouse.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR    Date       Author    Change made
---  ----------  ------  -----------------------------------------------


CSU_NAME:     HMI

FUNCTION:     f_cenh_handle_rmouse

PURPOSE:    This is the handler function f_o_r the right mouse button.  The
action is dependant on the current state of the FOV editor.

INPUT                                   SOURCE
-----                                   ------

OUTPUT                                   DESTINATION
------                                   -----------


CREATION DATE:     June 03 1993

===========================================================================*/
#include "cenh.h"

void f_cenh_handle_rmouse(void)
    /* Environment Definition HMI handle_rmouse function */
{
/*|

    determine which type of data is currently being edited
    SWITCH active_type DO
    |
    |   CASE c_clutter:
    |   |
    |   |   determine what the current edit mode is
    |   |   SWITCH edit_mode DO
    |   |   |
    |   |   |   CASE c_new: cancel the current polygon definition
    |   |   |   CASE c_add: cancel the current polygon definition
    |   |   |   |   USE f_cenh_edit_object(c_current,&t_cend_clutter_patch,patch_count,
    |   |   |   |   c_delete_element) TO delete the temporary patch from the CSS
    |   |   |   |   USE predraw_all_structs() TO redraw the FOV display with the new data
    |   |   |   |
    |   |   |   CASE c_modify:
    |   |   |   CASE c_modify_point:
    |   |   |   |   set edit_mode = c_no_op
    |   |   |   |
    |   |   |   CASE default:
    |   |   |   |   display FOV popup menu
    |   |   |   |
    |   |   ENDSWITCH
    |   |
    |   CASE c_discrete:
    |   CASE c_chaff:
    |   CASE c_cell:
    |   CASE c_jammer:
    |   CASE c_target:
    |   |   display FOV popup menu
    |   |
    ENDSWITCH
*/
}
```

```
/* end of cenh_handle_rmouse.c */
```

```
/*%%===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_init.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR    Date        Author    Change made
---  -----------  ------  -----------------------------------------------


CSU_NAME:      HMI

FUNCTION:      f_cenh_init

PURPOSE:    This function performs X-Windows and PHIGS initialization procedures
f_o_r the Environment Definition parameter entry task.



INPUT                                SOURCE
-----                                ------

OUTPUT                                DESTINATION
------                                -----------


CREATION DATE:     June 03 1993

============================================================================*/
#include "cenh.h"

void f_cenh_init(void)
     /* Environment Definition HMI Task */
{
/*|
     USE Env_Noise_Iface = f_Create_Env_Noise_Iface() TO create the external noise
     data entry widgets

     USE Env_Ionosphere_Iface = f_Create_Env_Ionosphere_Iface() TO create the external ionosphere
     data entry widgets

     USE Env_Atmoshpere_Iface = f_Create_Env_Atmoshpere_Iface() TO create the external atmoshpere
     data entry widgets

     USE Env_Background_Clutter_Iface = f_Create_Env_Background_Clutter_Iface() TO create the
     background clutter data entry widgets

     USE Env_Sea_Clutter_Iface = f_Create_Env_Sea_Clutter_Iface() TO create the
     sea clutter data entry widgets

     USE Env_Land_Clutter_Iface = f_Create_Env_Land_Clutter_Iface() TO create the
     land clutter data entry widgets

     USE Env_Clutter_Patch_Iface = f_Create_Env_Clutter_Patch_Iface() TO create the
     clutter data entry widgets

     USE Env_Target_Iface = f_Create_Env_Target_Iface() TO create the
     target data entry widgets

     USE Env_Jammer_Iface = f_Create_Env_Jammer_Iface() TO create the
     jammer data entry widgets

     USE Env_Chaff_Iface = f_Create_Env_Chaff_Iface() TO create the
     chaff bundle data entry widgets

     USE Env_Cell_Iface = f_Create_Env_Cell_Iface() TO create the
     precipitation cell data entry widgets

```

```
        USE Env_Discrete_Clutter_Iface = f_Create_Env_Discrete_Clutter_Iface() TO create the
        earth discrete clutter data entry widgets

        USE Env_Seed_Iface = f_Create_Env_Seed_Iface() TO create the
        initial seed data entry widgets

        create selection list with scrollbars for POV objects

        create field selections for time/dwell parameters

        perform PHIGS-SI initialisaion

        USE popen_xphigs() TO open PHIGS with the mask set to PXPHIGS_INFO_FLAGS_NO_MON

        USE popen_ws TO open a PHIGS workstation

        USE XtAddEventHandler (Main_Window, ExposureMask, FALSE, f_cenh_redraw, 0) TO
        register the EXPOSURE event handler

        USE delete_window_atom = XInternAtom (XtDisplay(Main_Window),"WM_DELETE_WINDOW",
        TRUE ) TO get the window managers window kill event

        USE XmAddWMProtocols(Main_Window, &delete_window_atom, 1) TO add the protocol

        USE XmSetWMProtocolHooks(Main_Window, delete_window_atom, NULL, NULL,
        f_cenh_close, NULL) TO register the close callback

        initialize the PHIGS color table f_o_r each data type to be displayed which
        allows the use of indirect color table lookup during traversal

        initialize display geometry and scaling from world to screen co-ords

        default the view such that it is at max. range

    */
    }

    /* end of cenh_init.c */
```

```
/*%%=========================================================================

SPD - Signal Processing Demonstrater Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_empty_fov.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR    Date        Author    Change made
---  -----------  ------  ------------------------------------------------


CSU_NAME:     HMI

FUNCTION:     f_cenh_empty_fov

PURPOSE:    This function will delete all the image elements in the
PHIGS CSS structure f_o_r the FOV display without destroying
the label markers within the structure.

INPUT                                SOURCE
-----                                ------

OUTPUT                                DESTINATION
------                                -----------


CREATION DATE:     June 03 1993

==========================================================================*/
#include "cenh.h"

void f_cenh_empty_fov(void)
     /* Environment Definition HMI close function */
{
/*|
     USE pdel_elems_labels(patch_start, patch_end) TO delete clutter patch elements

     USE pdel_elems_labels(discrete_start, discrete_end) TO delete discrete clutter  elements

     USE pdel_elems_labels(chaff_start, chaff_end) TO delete chaff bundle elements

     USE pdel_elems_labels(cell_start, cell_end) TO delete precipitation cell elements

     USE pdel_elems_labels(jammer_start, jammer_end) TO delete jammer elements

     USE pdel_elems_labels(target_start, target_end) TO delete target elements
*/
}

/* end of cenh_empty_fov.c */
```

```
/*%%===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_load.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR   Date         Author   Change made
---  -----------  ------  --------------------------------------------------


CSU_NAME:      HMI

FUNCTION:      f_cenh_load

PURPOSE:    This function will load Environment Definition data into the
local database and configure the PHIGS CSS structures f_o_r the
FOV display.




INPUT                              SOURCE
-----                              ------

OUTPUT                              DESTINATION
------                              -----------


CREATION DATE:     June 03 1993

===========================================================================*/
#include "cenh.h"

void f_cenh_load(char *file_name)
    /* Environment Definition HMI Task */
{
/*|

    read Environment Definition data from the input file and
    store it in the local database
    USE fopen() TO open the Environment Definition data file

    IF error on open THEN
    |   report file open error
<-------return
    ELSE
    |   USE xdrstdio_create TO initialize a record oriented XDR stream
    |   object (xdrs) in XDR_DECODE mode
    |   USE xdr_t_read_hmi TO read Environment Definition data file into t_cend_env_def struct
    |   IF error on read THEN
    |   |   report file read error
<-----------return
    |   ENDIF
    |   USE xdr_destroy TO destroy xdrs
    ENDIF

    create the PHIGS CSS f_o_r the FOV display

    USE f_cenh_empty_fov TO empty but not destroy the PHIGS structure

    set the FOV viewing transformation to the default max position

    clear the flag indicating "frozen" cursor location

    FOR (patch_count=0 IN patch_count<t_cend_env_def.patch_count) LOOP
    |   USE f_cenh_select_list(patch_count,c_add,&t_cend_env_def.patch[patch_count].loc)
    |   TO update the selection list
    |   USE f_cenh_edit_object(c_patch,&t_cend_env_def.patch[patch_count],patch_count,
     |   c_add_element) TO add a clutter patch to the CSS
     ENDLOOP

     FOR (discrete_count=0 IN discrete_count<t_cend_env_def.discrete_count) LOOP
     |   USE f_cenh_select_list(discrete_count,c_add,&t_cend_env_def.discrete[discrete_count].loc)
     |   TO update the selection list
     |   USE f_cenh_edit_object(c_discrete, t_cend_env_def.discrete[discrete_count],        USE
f_cenh_select_list(discrete_count,c_

add,&t_cend_env_def.discrete[discrete_count].loc)
     |   discrete_count,c_add_element) TO add discrete clutter to the CSS
     ENDLOOP

     FOR (chaff_count=0 IN chaff_count<t_cend_env_def.chaff_count) LOOP
     |   USE f_cenh_select_list(chaff_count,c_add,&t_cend_env_def.chaff[chaff_count].loc)
     |   TO update the selection list
     |   USE f_cenh_edit_object(c_chaff, t_cend_env_def.chaff[chaff_count],
     |   chaff_count,c_add_element) TO add a chaff bundle to the CSS
     ENDLOOP

     FOR (cell_count=0 IN cell_count<t_cend_env_def.cell_count) LOOP
     |   USE f_cenh_select_list(cell_count,c_add,&t_cend_env_def.cell[cell_count].loc)
     |   TO update the selection list
     |   USE f_cenh_edit_object(c_cell, t_cend_env_def.cell[cell_count],
     |   cell_count, c_add_element) TO add a precipitation cell to the CSS
     ENDLOOP

     FOR (jammer_count=0 IN jammer_count<t_cend_env_def.jammer_count) LOOP
     |   USE f_cenh_select_list(jammer_count,c_add,&t_cend_env_def.jammer[jammer_count].loc)
     |   TO update the selection list
     |   USE f_cenh_edit_object(c_jammer, t_cend_env_def.jammer[jammer_count],
     |   jammer_count, c_add_element) TO add a jammer to the CSS
     ENDLOOP

     FOR (target_count=0 IN target_count<t_cend_env_def.target_count) LOOP
     |   USE f_cenh_select_list(target_count,c_add,&t_cend_env_def.target[target_count].loc)
     |   TO update the selection list
     |   USE f_cenh_edit_object(c_target, t_cend_env_def.target[target_count],
     |   target_count, c_add_element) TO add a target to the CSS
     ENDLOOP

     USE f_cenh_edit_object() TO add the beam contour to the CSS at the T=0 position

     USE UxPutText() TO update the SSP lat/long. coordinates

     USE f_cenh_redraw() TO redraw the FOV display with the new data

*/
}

/* end of cenh_load.c */
```

```
/*%%=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_ld_radar.c

Identification: (#(@))

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR   Date        Author   Change made
---  ----------   ------  -------------------------------------------------


CSU_NAME:      HMI

FUNCTION:      f_cenh_ld_radar

PURPOSE:    This function will load the radar definition data from the
specified file and generate a new FOV display based on the new SSP.




INPUT                              SOURCE
-----                              ------

OUTPUT                              DESTINATION
------                              -----------


CREATION DATE:     June 03 1993

==========================================================================*/
#include "cenh.h"

void f_cenh_ld_radar(char *file_name)
    /* Environment Definition HMI Task */
{
/*|

    read radar Definition data from the input file and
    store it in the local database
    USE fopen() TO open the radar Definition data file

    IF error on open THEN
    |   report file open error
<--------return
    ELSE
    |   USE xdrstdio_create TO initialize a record oriented XDR stream
    |   object (xdrs) in XDR_DECODE mode
    |   USE xdr_t_read_hmi TO read radar Definition data file into t_cend_env_def struct
    |   IF error on read THEN
    |   |   report file read error
<------------return
    |   ENDIF
    |   USE xdr_destroy TO destroy xdrs
    ENDIF

*/
}

/* end of cenh_ld_radar.c */
```

```
/*%%========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_main.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR    Date       Author    Change made
---  ----------  ------  ---------------------------------------------------


CSU_NAME:     HMI

FUNCTION:     f_cenh_main

PURPOSE:    This task provides an X-Windows user interface f_o_r the
Environment Definition parameter entry.



INPUT                                SOURCE
-----                                ------

OUTPUT                               DESTINATION
------                               -----------


CREATION DATE:     June 03 1993

==========================================================================*/
#include "cenh.h"

f_cenh_main(int argc, char **argv)
    /* Environment Definition HMI Task */
{
/*|
    USE UxTopLevel = UxInitialize("hmi", &argc, argv) TO initialize X-Windows application
    UxTopLevel is a UxLib defined variable

    USE cenh_init() TO create X interfaces and initialize global data

    WHILE (FOREVER) LOOP
     |
     |   USE UxNextEvent(&event) TO get the next X-Event off the queue
     |
     |   USE UxDispatchEvent(&event) TO process the X-Event (state driven)
     |
     |   the callback routines being described are actually handled by
     |   UxDispatchEvent() they are explicity described here f_o_r the clarity
     |   of the design
     |   the callbacks are grouped according to the state of the HMI
     |   (eg. exclusive_grab of a sub-screen)
     |
     |   SWITCH CENH_State
     |    |
     |    |   CASE MAIN_FOV:
     |    |    |
     |    |    |   SWITCH Event
     |    |    |    |
     |    |    |    |   CASE POPUP.FILE.ENV.OPEN:
     |    |    |    |    |   display a file selection dialog
     |    |    |    |    |   USE f_cenh_load (&file_name) TO read envirnmnet data file into database
     |    |    |    |    |
     |    |    |    |   CASE POPUP.FILE.ENV.SAVE:
     |    |    |    |    |   display a file selection dialog
     |    |    |    |    |   USE f_cenh_save (&file_name) TO write database into file
     |    |    |    |    |
     |    |    |    |   CASE POPUP.FILE.RADAR.OPEN:
     |    |    |    |    |   display a file selection dialog
|   |   |   |   |   USE f_cenh_ld_radar (&file_name) TO read radar definition file into database
|   |   |   |   |   update SSP and FOV display
|   |   |   |   |
|   |   |   |   CASE POPUP.FILE.EXIT:
|   |   |   |   |   USE f_cenh_close() TO close the screen, save files etc. and exit
|   |   |   |   |
|   |   |   |   CASE POPUP.VIEW.MAX:
|   |   |   |   |   reset the viewing transform to give max field of view
|   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |   |
|   |   |   |   CASE POPUP.VIEW.CURRENT:
|   |   |   |   |   reset the viewing transform to give current FOV over entire window
|   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |   |
|   |   |   |   CASE POPUP.VIEW.ZOOM:
|   |   |   |   |   user indicates opposite corners of a rectangle f_o_r zoom region
|   |   |   |   |   set active_type=c_zoom
|   |   |   |   |   set edit_mode=c_new
|   |   |   |   |
|   |   |   |   CASE POPUP.CREATE:
|   |   |   |   |   set the edit_mode=c_new
|   |   |   |   |
|   |   |   |   CASE POPUP.DELETE:
|   |   |   |   |   set the edit_mode=c_delete
|   |   |   |   |
|   |   |   |   CASE POPUP_FOV.CREATE
|   |   |   |   |   set the edit_mode=c_new
|   |   |   |   |
|   |   |   |   CASE POPUP_FOV.DELETE
|   |   |   |   |   set the edit_mode=c_delete
|   |   |   |   |
|   |   |   |   CASE POPUP_FOV.VERTEX.ADD
|   |   |   |   |   set edit_mode=c_modify
|   |   |   |   |
|   |   |   |   CASE POPUP_FOV.VERTEX.DELETE
|   |   |   |   |   set the edit_mode=c_delete_vertex
|   |   |   |   |
|   |   |   |   CASE POPUP_FOV.COPY
|   |   |   |   |   make a copy of the next selected item
|   |   |   |   |   set edit_mode=c_copy
|   |   |   |   |
|   |   |   |   CASE POPUP_FOV.EDIT
|   |   |   |   |   set edit_mode=c_modify
|   |   |   |   |
|   |   |   |   CASE LEFT_MIDDLE_MOUSE:
|   |   |   |   |   when left and middle mouse are pressed simultaneously save
|   |   |   |   |   the current mouse location and mark that position with an "X" on the FOV
|   |   |   |   |   IF (cursor_frozen=FALSE) THEN
|   |   |   |   |   |   set cursor_frozen = TRUE
|   |   |   |   |   ENDIF
|   |   |   |   |
|   |   |   |   CASE LEFT_MOUSE:
|   |   |   |   |   when left mouse button pressed when cursor on FOV display
|   |   |   |   |   USE f_cenh_lmouse() TO process the appropriate action
|   |   |   |   |
|   |   |   |   CASE MIDDLE_MOUSE:
|   |   |   |   |   when middle mouse button pressed when cursor on FOV display
|   |   |   |   |   USE f_cenh_mmouse() TO process the appropriate action
|   |   |   |   |
|   |   |   |   CASE RIGHT_MOUSE:
|   |   |   |   |   when right mouse button pressed when cursor on FOV display
|   |   |   |   |   USE f_cenh_rmouse() TO process the appropriate action
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.ENV_NOISE
|   |   |   |   |   set data_status=c_unchanged
|   |   |   |   |   set active_type=c_noise
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE UxPopupInterface (Env_Noise_Iface,exclusive_grab) TO
|   |   |   |   |   display the external noise data entry screen
|   |   |   |   |
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.ENV_IONOSPHERE
|   |   |   |   |   set data_status=c_unchanged
|   |   |   |   |   set active_type=c_ionosphere
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE UxPopupInterface (Env_Ionosphere_Iface,exclusive_grab) TO
|   |   |   |   |   display the ionoshpere definition data entry screen
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.ENV_ATMOSPHERE
|   |   |   |   |   set data_status=c_unchanged
|   |   |   |   |   set active_type=c_atmosphere
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE UxPopupInterface (Env_Atmosphere_Iface,exclusive_grab) TO
|   |   |   |   |   display the atmoshpere definition data entry screen
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.ENV_BACKGROUND_CLUTTER:
|   |   |   |   |   set data_status=c_unchanged
|   |   |   |   |   set active_type=c_back_clutter
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE UxPopupInterface (Env_Background_Clutter_Iface,exclusive_grab) TO
|   |   |   |   |   display the background clutter definition data entry screen
|   |   |   |   |   IF (Clutter_Type==LAND) THEN
|   |   |   |   |   |   USE UxPopdownInterface (Sea_Clutter_Iface,no_grab) TO remove the
|   |   |   |   |   |   sea clutter wind direction data entry widget
|   |   |   |   |   |   USE UxPopupInterface (Land_Clutter_Iface,no_grab) TO display the
|   |   |   |   |   |   the land clutter wind speed data entry widget
|   |   |   |   |   ELSE
|   |   |   |   |   |   USE UxPopdownInterface (Land_Clutter_Iface,no_grab) TO remove the
|   |   |   |   |   |   the land clutter wind speed data entry widget
|   |   |   |   |   |   USE UxPopupInterface (Sea_Clutter_Iface,no_grab) TO display the
|   |   |   |   |   |   sea clutter wind direction data entry widget
|   |   |   |   |   ENDIF
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.ENV_CLUTTER_PATCH:
|   |   |   |   |   set data_status=c_unchanged
|   |   |   |   |   set active_type=c_clutter
|   |   |   |   |   set edit_mode=c_select
|   |   |   |   |   update fields with current values in database and last selected
|   |   |   |   |   clutter patch ID
|   |   |   |   |   USE f_cenh_select_list() TO highlight the current element
|   |   |   |   |   IF (Clutter_Type==LAND) THEN
|   |   |   |   |   |   USE UxPopdownInterface (Sea_Clutter_Iface,no_grab) TO remove the
|   |   |   |   |   |   sea clutter wind direction data entry widget
|   |   |   |   |   |   USE UxPopupInterface (Land_Clutter_Iface,no_grab) TO display the
|   |   |   |   |   |   the land clutter wind speed data entry widget
|   |   |   |   |   ELSE
|   |   |   |   |   |   USE UxPopdownInterface (Land_Clutter_Iface,no_grab) TO remove the
|   |   |   |   |   |   the land clutter wind speed data entry widget
|   |   |   |   |   |   USE UxPopupInterface (Sea_Clutter_Iface,no_grab) TO display the
|   |   |   |   |   |   sea clutter wind direction data entry widget
|   |   |   |   |   ENDIF
|   |   |   |   |   USE UxPopupInterface (Env_Clutter_Patch_Iface,exclusive_grab) TO
|   |   |   |   |   display the clutter patch definition data entry screen
|   |   |   |   |   allow user to perform interactive edit of clutter patches on FOV
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.ENV_TARGET:
|   |   |   |   |   set data_status=c_unchanged
|   |   |   |   |   set active_type=c_target
|   |   |   |   |   set edit_mode=c_select
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE f_cenh_select_list() TO highlight the current element
|   |   |   |   |   USE UxPopupInterface (Env_Target_Iface,exclusive_grab) TO
|   |   |   |   |   display the target definition data entry screen and
|   |   |   |   |   allow user to perform interactive edit of targets on FOV
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.ENV_JAMMER:
|   |   |   |   |   set data_status=c_unchanged
|   |   |   |   |   set active_type=c_jammer
|   |   |   |   |   set edit_mode=c_select
|   |   |   |   |   USE f_cenh_select_list() TO highlight the current element
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE UxPopupInterface (Env_Jammer_Iface,exclusive_grab) TO
|   |   |   |   |   display the jammer definition data entry screen
|   |   |   |   |   allow user to perform interactive edit of jammer positions on FOV
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.ENV_CHAFF:
|   |   |   |   |   set data_status=c_unchanged
|   |   |   |   |   set active_type=c_chaff
|   |   |   |   |   set edit_mode=c_select
|   |   |   |   |   USE f_cenh_select_list() TO highlight the current element
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE UxPopupInterface (Env_Chaff_Iface,exclusive_grab) TO
|   |   |   |   |   display the chaff bundle definition data entry screen
|   |   |   |   |   allow user to perform interactive edit of chaff positions on FOV
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.ENV_CELL:
|   |   |   |   |   set data_status=c_unchanged
|   |   |   |   |   set active_type=c_cell
|   |   |   |   |   set edit_mode=c_select
|   |   |   |   |   USE f_cenh_select_list() TO highlight the current element
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE UxPopupInterface (Env_Cell_Iface,exclusive_grab) TO
|   |   |   |   |   display the precipitation cell definition data entry screen
|   |   |   |   |   allow user to perform interactive edit of precipitation
|   |   |   |   |   cell positions on FOV
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.ENV_DISCRETE_CLUTTER:
|   |   |   |   |   set data_status=c_unchanged
|   |   |   |   |   set active_type=c_discrete
|   |   |   |   |   USE f_cenh_select_list() TO highlight the current element
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE UxPopupInterface (Env_Discrete_Clutter_Iface,exclusive_grab) TO
|   |   |   |   |   display the earth discrete clutter definition data entry screen
|   |   |   |   |   allow user to perform interactive edit of clutter patches on FOV
|   |   |   |   |
|   |   |   |   CASE ENV_SEED:
|   |   |   |   |   set data_status=c_unchanged
|   |   |   |   |   set active_type=c_seed
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE UxPopupInterface (Env_Seed_Iface,exclusive_grab) TO
|   |   |   |   |   display the seed definition data entry screen
|   |   |   |   |
|   |   |   ENDSWITCH
|   |   |
|   |   CASE TIME_DWELL:
|   |   |   generate prediction/extrapolation of data based on time/dwell
|   |   |   information f_o_r currently selected object and display that data
|   |   |   in a scrolled window
|   |   |   USE f_cenh_edit_object() TO update the beam position on the FOV display
|   |   |
|   |   CASE ENV_NOISE:
|   |   |   SWITCH Event
|   |   |   |
|   |   |   |   CASE RADIO_BUTTON.MODEL.ON:
|   |   |   |   |
|   |   |   |   CASE RADIO_BUTTON.MODEL.OFF:
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.OK
|   |   |   |   |   copy user data into database
|   |   |   |   |   USE UxPopdownInterface(Env_Noise_Iface) TO close the
|   |   |   |   |   external noise data entry screen
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.CANCEL
|   |   |   |   |   USE UxPopdownInterface(Env_Noise_Iface) TO close the
|   |   |   |   |   external noise data entry screen
|   |   |   |   |
|   |   |   ENDSWITCH
|   |   |
|   |   CASE ENV_IONOSPHERE:
|   |   |   SWITCH Event
|   |   |   |
|   |   |   |   CASE RADIO_BUTTON.MODEL.ON:
|   |   |   |   |
|   |   |   |   |   RADIO_BUTTON.MODEL.OFF:
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.ELECTRON_DENSITY:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.ION_ALTITUDE:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   update  SCROLLBAR.ION_ALTITUDE to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.ION_ALTITUDE:
|   |   |   |   |   update FIELD_ENTRY.ION_ALTITUDE to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.SCALE_HEIGHT:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   update SCROLLBAR.SCALE_HEIGHT to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.SCALE_HEIGHT:
|   |   |   |   |   update FIELD_ENTRY.SCALE_HEIGHT to indicate new value
|   |   |   |   |
|   |   |   |   |
|   |   |   |   CASE OPTION_MENU.SCINTILLATION(0.0):
|   |   |   |   CASE OPTION_MENU.SCINTILLATION(0.1):
|   |   |   |   CASE OPTION_MENU.SCINTILLATION(0.2):
|   |   |   |   CASE OPTION_MENU.SCINTILLATION(0.3):
|   |   |   |   CASE OPTION_MENU.SCINTILLATION(0.4):
|   |   |   |   CASE OPTION_MENU.SCINTILLATION(0.5):
|   |   |   |   |   store selected value
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.OK
|   |   |   |   |   copy user data into database
|   |   |   |   |   USE UxPopdownInterface(Env_Ionosphere_Iface) TO close the
|   |   |   |   |   ionosphere definition data entry screen
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.CANCEL
|   |   |   |   |   USE UxPopdownInterface(Env_Ionosphere_Iface) TO close the
|   |   |   |   |   ionosphere definition data entry screen
|   |   |   |   |
|   |   |   ENDSWITCH
|   |   |
|   |   CASE ENV_ATMOSPHERE:
|   |   |   SWITCH Event
|   |   |   |
|   |   |   |   CASE RADIO_BUTTON.LOWER_ATM.ON:
|   |   |   |   |
|   |   |   |   CASE RADIO_BUTTON.LOWER_ATM.OFF:
|   |   |   |   |
|   |   |   |   CASE OPTION_MENU.HUMIDITY.CCIR_AVERAGE:
|   |   |   |   CASE OPTION_MENU.HUMIDITY.100_PERCENT:
|   |   |   |   |
|   |   |   |   CASE TOGGLEBUTTON.HUMIDITY.OFF:
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.OK
|   |   |   |   |   copy user data into database
|   |   |   |   |   USE UxPopdownInterface(Env_Atmosphere_Iface) TO close the
|   |   |   |   |   atmosphere definition data entry screen
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.CANCEL
|   |   |   |   |   USE UxPopdownInterface(Env_Atmosphere_Iface) TO close the
|   |   |   |   |   atmosphere definition data entry screen
|   |   |   |   |
|   |   |   ENDSWITCH
|   |   |
|   |   CASE ENV_CLUTTER_PATCH:
|   |   |
|   |   |   SWITCH Event
|   |   |   |
|   |   |   |   CASE FIELD_ENTRY.PATCH_ID:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   IF (Entry_Valid==VALID) THEN
|   |   |   |   |   |   update fields with current values in database
|   |   |   |   |   ENDIF
|   |   |   |   |
|   |   |   |   CASE OPTION_MENU.TYPE.LAND:
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE UxPopdownInterface (Sea_Clutter_Iface,no_grab) TO remove the
|   |   |   |   |   sea clutter wind direction data entry widget
|   |   |   |   |   USE UxPopupInterface (Land_Clutter_Iface,no_grab) TO display the
|   |   |   |   |   the land clutter wind speed data entry widget
|   |   |   |   |
|   |   |   |   CASE OPTION_MENU.TYPE.SEA:
|   |   |   |   |   update fields with current values in database
|   |   |   |   |   USE UxPopdownInterface (Land_Clutter_Iface,no_grab) TO remove the
|   |   |   |   |   the land clutter wind speed data entry widget
|   |   |   |   |   USE UxPopupInterface (Sea_Clutter_Iface,no_grab) TO display the
|   |   |   |   |   sea clutter wind direction data entry widget
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.LAND.SPEED:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   update SCROLLBAR.LAND.SPEED to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.LAND.SPEED:
|   |   |   |   |   update SCROLLBAR.LAND.SPEED to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.SEA.DIRECTION:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   update SCROLLBAR.SEA.DIRECTION to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.SEA.DIRECTION:
|   |   |   |   |   update SCROLLBAR.SEA.DIRECTION to indicate new value
|   |   |   |   |
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(0):
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(1):
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(2):
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(3):
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(4):
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(5):
|   |   |   |   |
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.OK
|   |   |   |   |   copy user data into database
|   |   |   |   |   IF (new patch was defined) THEN
|   |   |   |   |   |   USE
f_cenh_edit_object(c_patch,&t_cend_env_def.patch[patch_count],patch_count,
|   |   |   |   |   |   c_add_element) TO add a clutter patch to the CSS
|   |   |   |   |   ELSE
|   |   |   |   |   |   USE
f_cenh_edit_object(c_patch,&t_cend_env_def.patch[patch_count],patch_count,
|   |   |   |   |   |   c_replace_element) TO add a clutter patch to the CSS
|   |   |   |   |   ENDIF
|   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |   |   USE UxPopdownInterface(Env_Clutter_Patch_Iface) TO close the
|   |   |   |   |   clutter definition data entry screen
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.APPLY
|   |   |   |   |   copy user data into database f_o_r current clutter patch
|   |   |   |   |   IF (new patch was defined) THEN
|   |   |   |   |   |   USE
f_cenh_edit_object(c_patch,&t_cend_env_def.patch[patch_count],patch_count,
|   |   |   |   |   |   c_add_element) TO add a clutter patch to the CSS
|   |   |   |   |   ELSE
|   |   |   |   |   |   USE
f_cenh_edit_object(c_patch,&t_cend_env_def.patch[patch_count],patch_count,
|   |   |   |   |   |   c_edit_element) TO add a clutter patch to the CSS
|   |   |   |   |   ENDIF
|   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.CANCEL
|   |   |   |   |   USE UxPopdownInterface(Env_Clutter_Patch_Iface) TO close the
|   |   |   |   |   clutter definition data entry screen
|   |   |   |   |
|   |   |   ENDSWITCH
|   |   |
|   |   CASE ENV_BACKGROUND_CLUTTER:
|   |   |
|   |   |   SWITCH Event
|   |   |   |
|   |   |   |   CASE RADIO_BUTTON.CLUTTER.ON:
|   |   |   |   |
|   |   |   |   CASE RADIO_BUTTON.CLUTTER.OFF:
|   |   |   |   |
|   |   |   |   CASE OPTION_MENU.TYPE.LAND:
|   |   |   |   |   USE UxPopdownInterface (Sea_Clutter_Iface,no_grab) TO remove the
|   |   |   |   |   sea clutter wind direction data entry widget
|   |   |   |   |   USE UxPopupInterface (Land_Clutter_Iface,no_grab) TO display the
|   |   |   |   |   the land clutter wind speed data entry widget
|   |   |   |   |
|   |   |   |   CASE OPTION_MENU.TYPE.SEA:
|   |   |   |   |   USE UxPopdownInterface (Land_Clutter_Iface,no_grab) TO remove the
|   |   |   |   |   the land clutter wind speed data entry widget
|   |   |   |   |   USE UxPopupInterface (Sea_Clutter_Iface,no_grab) TO display the
|   |   |   |   |   sea clutter wind direction data entry widget
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.LAND.SPEED:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   update SCROLLBAR.LAND.SPEED to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.LAND.SPEED:
|   |   |   |   |   update SCROLLBAR.LAND.SPEED to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.SEA.DIRECTION:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   update SCROLLBAR.SEA.DIRECTION to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.SEA.DIRECTION:
|   |   |   |   |   update SCROLLBAR.SEA.DIRECTION to indicate new value
|   |   |   |   |
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(0):
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(1):
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(2):
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(3):
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(4):
|   |   |   |   CASE OPTION_ENTRY.SEA.STATE(5):
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.OK
|   |   |   |   |   copy user data into database
|   |   |   |   |   USE UxPopdownInterface(Env_Background_Clutter_Iface) TO close the
|   |   |   |   |   background clutter definition data entry screen
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.CANCEL
|   |   |   |   |   USE UxPopdownInterface(Env_Background_Clutter_Iface) TO close the
|   |   |   |   |   background clutter definition data entry screen
|   |   |   |   |
|   |   |   |   |
|   |   |   ENDSWITCH
|   |   |
|   |   CASE ENV_TARGET:
|   |   |   SWITCH Event
|   |   |   |
|   |   |   |   CASE FIELD_ENTRY.TARGET_ID:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   IF (Entry_Valid==VALID) THEN
|   |   |   |   |   |   update fields with current values in database f_o_r specified target
|   |   |   |   |   |   highlight the target on the FOV display
|   |   |   |   |   ENDIF
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.LAT:
```

```
|     |     |     |     |   |   USE f_cenh_validate_field() TO perform range check on field entry
|     |     |     |     |   |   update SCROLLBAR.LAT to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE SCROLLBAR.LAT:
|     |     |     |     |   |   update FIELD_ENTRY.LAT to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE FIELD_ENTRY.LONG:
|     |     |     |     |   |   USE f_cenh_validate_field() TO perform range check on field entry
|     |     |     |     |   |   update SCROLLBAR.LONG to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE SCROLLBAR.LONG:
|     |     |     |     |   |   update FIELD_ENTRY.LONG to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE FIELD_ENTRY.HEADING:
|     |     |     |     |   |   USE f_cenh_validate_field() TO perform range check on field entry
|     |     |     |     |   |   update SCROLLBAR.HEADING to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE SCROLLBAR.HEADING:
|     |     |     |     |   |   update FIELD_ENTRY.HEADING to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE FIELD_ENTRY.TARGET_ALTITUDE:
|     |     |     |     |   |   USE f_cenh_validate_field() TO perform range check on field entry
|     |     |     |     |   |   update SCROLLBAR.TARGET_ALTITUDE to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE SCROLLBAR.TARGET_ALTITUDE:
|     |     |     |     |   |   update FIELD_ENTRY.TARGET_ALTITUDE to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE FIELD_ENTRY.TARGET_SPEED:
|     |     |     |     |   |   USE f_cenh_validate_field() TO perform range check on field entry
|     |     |     |     |   |   update SCROLLBAR.TARGET_SPEED to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE SCROLLBAR.TARGET_SPEED:
|     |     |     |     |   |   update FIELD_ENTRY.TARGET_SPEED to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE FIELD_ENTRY.SPECTRAL_WIDTH:
|     |     |     |     |   |   USE f_cenh_validate_field() TO perform range check on field entry
|     |     |     |     |   |   update SCROLLBAR.SPECTRAL_WIDTH to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE SCROLLBAR.SPECTRAL_WIDTH:
|     |     |     |     |   |   update FIELD_ENTRY.SPECTRAL_WIDTH to indicate new value
|     |     |     |     |   |
|     |     |     |     |   |   RADIO.CAPTURE.ON:
|     |     |     |     |   |
|     |     |     |     |   |   RADIO.CAPTURE.OFF:
|     |     |     |     |   |
|     |     |     |     |   |   OPTION_MENU.RCS_PROFILE.PROVIDED_TABLE:
|     |     |     |     |   |
|     |     |     |     |   |   OPTION_MENU.RCS_PROFILE.LOAD_FROM_FILE:
|     |     |     |     |   |   USE File_Selection Widget TO select a file
|     |     |     |     |   |   USE cenh_loadrcs() TO load the parameters into the database
|     |     |     |     |   |
|     |     |     |     |   CASE FIELD_ENTRY.TARGET_SEED:
|     |     |     |     |   |   USE f_cenh_validate_field(SEED,&field_entry) TO perform range check on field entry
|     |     |     |     |   |
|     |     |     |     |   CASE FIELD_ENTRY.RCS:
|     |     |     |     |   |   USE f_cenh_validate_field() TO perform range check on field entry
|     |     |     |     |   |   update SCROLLBAR.RCS to indicate new value
|     |     |     |     |   |
|     |     |     |     |   CASE SCROLLBAR.RCS:
|     |     |     |     |   |   update FIELD_ENTRY.RCS to indicate new value
|     |     |     |     |   |
|     |     |     |     |   |   PUSHBUTTON.TVPT:
|     |     |     |     |   |   update the TVPT (Time Varying parameter Table) screen from database
|     |     |     |     |   |   USE UxPopupInterface(TVPT_Iface,no_grab) TO display data
|     |     |     |     |   |
|     |     |     |     |   |   PUSHBUTTON.OK
|     |     |     |     |   |   copy user data into database
|     |     |     |     |   |   IF (new target was defined) THEN
|     |     |     |     |   |   |   USE f_cenh_edit_object(c_target,&t_cend_env_def.target[target_count],
|     |     |     |     |   |   |   target_count, c_add_element) TO add a target to the CSS
|     |     |     |     |   |   ELSE
|     |     |     |     |   |   |   USE f_cenh_edit_object(c_target,&t_cend_env_def.target[target_count],
|     |     |     |     |   |   |   target_count, c_replace_element) TO edit a target to the CSS
|     |     |     |     |   |   ENDIF
|     |     |     |     |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|     |     |     |     |   |   USE UxPopdownInterface(Env_Target_Iface) TO close the
|     |     |     |     |   |   target definition data entry screen
|     |     |     |     |   |
|     |     |     |     |   |   PUSHBUTTON.CANCEL
|     |     |     |     |   |   USE UxPopdownInterface(Env_Target_Iface) TO close the
|     |     |     |     |   |   target definition data entry screen
|     |     |     |     |   |
|     |     |     |     |   |   PUSHBUTTON.APPLY
|   |   |   |   |   copy user data into database f_o_r current target
|   |   |   |   |   IF (new target was defined) THEN
|   |   |   |   |   |   USE f_cenh_edit_object(c_target,&t_cend_env_def.target[target_count],
|   |   |   |   |   |   target_count, c_add_element) TO add a target to the CSS
|   |   |   |   |   ELSE
|   |   |   |   |   |   USE f_cenh_edit_object(c_target,&t_cend_env_def.target[target_count],
|   |   |   |   |   |   target_count, c_replace_element) TO edit a target to the CSS
|   |   |   |   |   ENDIF
|   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |   |
|   |   |   ENDSWITCH
|   |   |
|   |   |   ENV_JAMMER:
|   |   |   SWITCH Event
|   |   |   |
|   |   |   |   CASE FIELD_ENTRY.JAMMER_ID:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   update data fields with jammer parameters f_o_r specified ID
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.LAT:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   update SCROLLBAR.LAT to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.LAT:
|   |   |   |   |   update FIELD_ENTRY.LAT to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.LONG:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   update SCROLLBAR.LONG to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.LONG:
|   |   |   |   |   update FIELD_ENTRY.LONG to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.HEADING:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |   update SCROLLBAR.HEADING to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.HEADING:
|   |   |   |   |   update FIELD_ENTRY.HEADING to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.ALTITUDE:
|   |   |   |   |   USE f_cenh_validate_field(ALTITUDE,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   update SCROLLBAR.ALTITUDE to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.ALTITUDE:
|   |   |   |   |   update FIELD_ENTRY.ALTITUDE to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.JAMMER_SPEED:
|   |   |   |   |   USE f_cenh_validate_field(JAMMER_SPEED,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   update SCROLLBAR.JAMMER_SPEED to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.JAMMER_SPEED:
|   |   |   |   |   update FIELD_ENTRY.JAMMER_SPEED to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.JAMMER_SEED:
|   |   |   |   |   USE f_cenh_validate_field(SEED,&field_entry) TO perform range check on field
entry
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.ERP:
|   |   |   |   |   ERP is always displayed
|   |   |   |   |   USE f_cenh_validate_field(ERP,&field_entry) TO perform range check on field entry
|   |   |   |   |   update SCROLLBAR.ERP to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.ERP:
|   |   |   |   |   ERP is always displayed
|   |   |   |   |   update FIELD_ENTRY.ERP to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.START_TIME:
|   |   |   |   |   TIME is always displayed
|   |   |   |   |   USE f_cenh_validate_field(START_TIME,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   update SCROLLBAR.START_TIME to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.START_TIME:
|   |   |   |   |   TIME is always displayed
|   |   |   |   |   update FIELD_ENTRY.START_TIME to indicate new value
|   |   |   |   |
|   |   |   |   CASE OPTION_MENU.JAMMER_TYPE.BARRAGE:
|   |   |   |   |   allow CENTER_FREQUENCY selection
|   |   |   |   |   allow BANDWIDTH selection
|   |   |   |   |   allow STROBE_FREQUENCY selection
|   |   |   |   |   allow DUTY_CYCLE selection
|    |    |    |    |
|    |    |    |    CASE OPTION_MENU.JAMMER_TYPE.SWEPT:
|    |    |    |    |    allow MID_SWEEP_FREQUENCY selection
|    |    |    |    |    allow INSTANTANEOUS_BANDWIDTH selection
|    |    |    |    |    allow SWEPT_BANDWIDTH selection
|    |    |    |    |    allow SWEEP_RATE selection
|    |    |    |    |    allow STROBE_FREQUENCY selection
|    |    |    |    |    allow DUTY_CYCLE selection
|    |    |    |    |
|    |    |    |    CASE OPTION_MENU.JAMMER_TYPE.FOLLOWER:
|    |    |    |    |    allow OFFSET_FREQUENCY selection
|    |    |    |    |    allow INSTANTANEOUS_BANDWIDTH selection
|    |    |    |    |    allow PROCESSING_DELAY selection
|    |    |    |    |    allow STROBE_FREQUENCY selection
|    |    |    |    |    allow DUTY_CYCLE selection
|    |    |    |    |
|    |    |    |    CASE OPTION_MENU.JAMMER_TYPE.DECEPTION:
|    |    |    |    |    false target editing
|    |    |    |    |    allow FALSE_TARGETS field entry
|    |    |    |    |    allow DECEPTION_METHOD selection
|    |    |    |    |    allow RSS_GENERATED selection
|    |    |    |    |    this determines whether to generate a selection list f_o_r false targets
|    |    |    |    |
|    |    |    |    CASE FIELD_ENTRY.BANDWIDTH:
|    |    |    |    |    USE f_cenh_validate_field(BANDWIDTH,&field_entry) TO perform range
|    |    |    |    |    check on field entry
|    |    |    |    |    update SCROLLBAR.BANDWIDTH to indicate new value
|    |    |    |    |
|    |    |    |    CASE SCROLLBAR.BANDWIDTH:
|    |    |    |    |    update FIELD_ENTRY.BANDWIDTH to indicate new value
|    |    |    |    |
|    |    |    |    CASE FIELD_ENTRY.CENTRE_FREQUENCY:
|    |    |    |    |    USE f_cenh_validate_field(CENTRE_FREQUENCY,&field_entry) TO perform range
|    |    |    |    |    check on field entry
|    |    |    |    |    update SCROLLBAR.CENTRE_FREQUENCY to indicate new value
|    |    |    |    |
|    |    |    |    CASE SCROLLBAR.CENTRE_FREQUENCY:
|    |    |    |    |    update FIELD_ENTRY.CENTRE_FREQUENCY to indicate new value
|    |    |    |    |
|    |    |    |    CASE FIELD_ENTRY.STROBE_FREQUENCY:
|    |    |    |    |    USE f_cenh_validate_field(STROBE_FREQUENCY,&field_entry) TO perform
|    |    |    |    |    range check on field entry
|    |    |    |    |    update SCROLLBAR.STROBE_FREQUENCY to indicate new value
|    |    |    |    |
|    |    |    |    CASE SCROLLBAR.STROBE_FREQUENCY:
|    |    |    |    |    update FIELD_ENTRY.STROBE_FREQUENCY to indicate new value
|    |    |    |    |
|    |    |    |    CASE FIELD_ENTRY.DUTY_CYCLE:
|    |    |    |    |    USE f_cenh_validate_field(DUTY_CYCLE,&field_entry) TO perform range check on
field entry
|    |    |    |    |    update SCROLLBAR.DUTY_CYCLE to indicate new value
|    |    |    |    |
|    |    |    |    CASE SCROLLBAR.DUTY_CYCLE:
|    |    |    |    |    update FIELD_ENTRY.DUTY_CYCLE to indicate new value
|    |    |    |    |
|    |    |    |    CASE FIELD_ENTRY.SWEEP_FREQUENCY:
|    |    |    |    |    USE f_cenh_validate_field(SWEEP_FREQUENCY,&field_entry) TO perform
|    |    |    |    |    range check on field entry
|    |    |    |    |    update SCROLLBAR.SWEEP_FREQUENCY to indicate new value
|    |    |    |    |
|    |    |    |    CASE SCROLLBAR.SWEEP_FREQUENCY:
|    |    |    |    |    update FIELD_ENTRY.SWEEP_FREQUENCY to indicate new value
|    |    |    |    |
|    |    |    |    CASE FIELD_ENTRY.PROCESSING_DELAY:
|    |    |    |    |    USE f_cenh_validate_field(PROCESSING_DELAY,&field_entry) TO perform range
|    |    |    |    |    check on field entry
|    |    |    |    |    update SCROLLBAR.PROCESSING_DELAY to indicate new value
|    |    |    |    |
|    |    |    |    CASE SCROLLBAR.PROCESSING_DELAY:
|    |    |    |    |    update FIELD_ENTRY.PROCESSING_DELAY to indicate new value
|    |    |    |    |
|    |    |    |    |
|    |    |    |    CASE FIELD_ENTRY.FALSE_TARGET:
|    |    |    |    |    USE f_cenh_validate_field(FALSE_TARGET,&field_entry) TO perform range check on
field entry
|    |    |    |    |    number of false targets
|    |    |    |    |
|    |    |    |    |    RADIO_BUTTON.RSS_GENERATED.ON:
|    |    |    |    |    remove the selection list of false targets
|    |    |    |    |
|    |    |    |    |    RADIO_BUTTON.RSS_GENERATED.OFF:
|    |    |    |    |    display SELECTION_LIST.FALSE_TARGET of false targets (based on
|    |    |    |    |    number of false targets)
|    |    |    |    |
|    |    |    |    |    SELECTION_LIST.FALSE_TARGET.RANGE_OFFSET:
```

```
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |
|   |   |   |   |
|   |   |   |   |   SELECTION_LIST.FALSE_TARGET.VELOCITY_OFFSET:
|   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.TVPT:
|   |   |   |   |   update the TVPT (Time Varying Parameter Table) screen from database
|   |   |   |   |   USE UxPopupInterface(TVPT_Iface,no_grab) TO display data
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.OK
|   |   |   |   |   copy user data into database
|   |   |   |   |   IF (new jammer was defined) THEN
|   |   |   |   |   |   USE f_cenh_edit_object(c_target,&t_cend_env_def.jammer[jammer_count],
|   |   |   |   |   |   jammer_count, c_add_element) TO add a jammer to the CSS
|   |   |   |   |   ELSE
|   |   |   |   |   |   USE f_cenh_edit_object(c_target,&t_cend_env_def.jammer[jammer_count],
|   |   |   |   |   |   jammer_count, c_replace_element) TO edit a jammer in the CSS
|   |   |   |   |   ENDIF
|   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |   |   USE UxPopdownInterface(Env_Target_Iface) TO close the
|   |   |   |   |   target definition data entry screen
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.CANCEL
|   |   |   |   |   USE UxPopdownInterface(Env_Target_Iface) TO close the
|   |   |   |   |   target definition data entry screen
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.APPLY
|   |   |   |   |   copy user data into database f_o_r current target
|   |   |   |   |   IF (new jammer was defined) THEN
|   |   |   |   |   |   USE f_cenh_edit_object(c_jammer,&t_cend_env_def.jammer[jammer_count],
|   |   |   |   |   |   jammer_count, c_add_element) TO add a jammer to the CSS
|   |   |   |   |   ELSE
|   |   |   |   |   |   USE f_cenh_edit_object(c_jammer,&t_cend_env_def.jammer[jammer_count],
|   |   |   |   |   |   jammer_count, c_replace_element) TO edit a jammer in the CSS
|   |   |   |   |   ENDIF
|   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |   |
|   |   |   ENDSWITCH
|   |   |
|   |   |   ENV_CHAFF:
|   |   |   SWITCH Event
|   |   |   |
|   |   |   |   CASE FIELD_ENTRY.CHAFF_BUNDLE_ID:
|   |   |   |   |   USE f_cenh_validate_field(CHAFF_BUNDLE_ID,&field_entry) TO perform
|   |   |   |   |   range check on field entry
|   |   |   |   |   IF (Entry_Valid==VALID) THEN
|   |   |   |   |   |   update fields with current values in database
|   |   |   |   |   ENDIF
|   |   |   |   |   highlight the specified chaff bundle on the FOV display
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.LAT:
|   |   |   |   |   USE f_cenh_validate_field(LATITUDE,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   update SCROLLBAR.LAT to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.LAT:
|   |   |   |   |   update FIELD_ENTRY.LAT to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.LONG:
|   |   |   |   |   USE f_cenh_validate_field(LONGITUDE,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   update SCROLLBAR.LONG to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.LONG:
|   |   |   |   |   update FIELD_ENTRY.LONG to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.ALTITUDE:
|   |   |   |   |   USE f_cenh_validate_field(ALTITUDE,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   update SCROLLBAR.ALTITUDE to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.ALTITUDE:
|   |   |   |   |   update FIELD_ENTRY.ALTITUDE to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.WEIGHT:
|   |   |   |   |   USE f_cenh_validate_field(WEIGHT,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   update SCROLLBAR.WEIGHT to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.WEIGHT:
|   |   |   |   |   update FIELD_ENTRY.WEIGHT to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.STAGGER_TUNE:
|   |   |   |   |   USE f_cenh_validate_field(STAGGER_TUNE,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   update SCROLLBAR.STAGGER_TUNE to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.STAGGER_TUNE:
|   |   |   |   |   update FIELD_ENTRY.STAGGER_TUNE to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.DISPERSION:
|   |   |   |   |   USE f_cenh_validate_field(DISPERSION,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   update SCROLLBAR.DISPERSION to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.DISPERSION:
|   |   |   |   |   update FIELD_ENTRY.DISPERSION to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.WIND_SPEED:
|   |   |   |   |   USE f_cenh_validate_field(SPEED,&field_entry) TO perform range check on field entry
|   |   |   |   |   update SCROLLBAR.WIND_SPEED to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.WIND_SPEED:
|   |   |   |   |   update FIELD_ENTRY.WIND_SPEED to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.WIND_DIRECTION:
|   |   |   |   |   USE f_cenh_validate_field(DIRECTION,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   update SCROLLBAR.WIND_DIRECTION to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.WIND_DIRECTION:
|   |   |   |   |   update FIELD_ENTRY.WIND_DIRECTION to indicate new value
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.OK
|   |   |   |   |   copy user data into database
|   |   |   |   |   IF (new chaff was defined) THEN
|   |   |   |   |   |   USE f_cenh_edit_object(c_chaff,&t_cend_env_def.chaff[chaff_count],
|   |   |   |   |   |   chaff_count, c_add_element) TO add a chaff bundle to the CSS
|   |   |   |   |   ELSE
|   |   |   |   |   |   USE f_cenh_edit_object(c_chaff,&t_cend_env_def.chaff[chaff_count],
|   |   |   |   |   |   chaff_count, c_replace_element) TO edit a chaff bundle in the CSS
|   |   |   |   |   ENDIF
|   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |   |   USE UxPopdownInterface(Env_Chaff_Iface) TO close the
|   |   |   |   |   chaff bundle definition data entry screen
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.CANCEL
|   |   |   |   |   USE UxPopdownInterface(Env_Chaff_Iface) TO close the
|   |   |   |   |   chaff bundle definition data entry screen
|   |   |   |   |
|   |   |   |   CASE PUSHBUTTON.APPLY
|   |   |   |   |   copy user data into database f_o_r current chaff bundle
|   |   |   |   |   IF (new chaff was defined) THEN
|   |   |   |   |   |   USE f_cenh_edit_object(c_chaff,&t_cend_env_def.chaff[chaff_count],
|   |   |   |   |   |   chaff_count, c_add_element) TO add a chaff bundle to the CSS
|   |   |   |   |   ELSE
|   |   |   |   |   |   USE f_cenh_edit_object(c_chaff,&t_cend_env_def.chaff[chaff_count],
|   |   |   |   |   |   chaff_count, c_replace_element) TO edit a chaff bundle in the CSS
|   |   |   |   |   ENDIF
|   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
|   |   |   |   |
|   |   |   ENDSWITCH
|   |   |
|   |   |   ENV_CELL:
|   |   |   SWITCH Event
|   |   |   |
|   |   |   |   CASE FIELD_ENTRY.CELL_ID:
|   |   |   |   |   USE f_cenh_validate_field(CELL_ID,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   IF (Entry_Valid==VALID) THEN
|   |   |   |   |   |   update fields with current values in database
|   |   |   |   |   ENDIF
|   |   |   |   |   highlight the specified precipitation cell on the FOV display
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.LAT:
|   |   |   |   |   USE f_cenh_validate_field(LATITUDE,&field_entry) TO perform range
|   |   |   |   |   check on field entry
|   |   |   |   |   update SCROLLBAR.LAT to indicate new value
|   |   |   |   |
|   |   |   |   CASE SCROLLBAR.LAT:
|   |   |   |   |   update FIELD_ENTRY.LAT to indicate new value
|   |   |   |   |
|   |   |   |   CASE FIELD_ENTRY.LONG:
|   |   |   |   |   USE f_cenh_validate_field(LONGITUDE,&field_entry) TO perform range
|   |   |   |   |   check on field entry
| | | | |   update SCROLLBAR.LONG to indicate new value
| | | | |
| | | | CASE SCROLLBAR.LONG:
| | | | |   update FIELD_ENTRY.LONG to indicate new value
| | | | |
| | | | CASE OPTION_MENU.WATER_STATE.RAIN:
| | | | CASE OPTION_MENU.WATER_STATE.SNOW:
| | | | CASE OPTION_MENU.WATER_STATE.FOG:
| | | | CASE OPTION_MENU.WATER_STATE.CLOUD:
| | | | |   update field entries to indicate appropriate values f_o_r water state
| | | | |
| | | | CASE FIELD_ENTRY.CIRCULAR_RADIUS:
| | | | |   USE f_cenh_validate_field(CIRCULAR_RADIUS,&field_entry) TO perform range
| | | | |   check on field entry
| | | | |   update SCROLLBAR.CIRCULAR_RADIUS to indicate new value
| | | | |
| | | | CASE SCROLLBAR.CIRCULAR_RADIUS:
| | | | |   update FIELD_ENTRY.CIRCULAR_RADIUS to indicate new value
| | | | |
| | | | CASE FIELD_ENTRY.LOWER_ALTITUDE:
| | | | |   USE f_cenh_validate_field(LOWER_ALTITUDE,&field_entry) TO perform range
| | | | |   check on field entry
| | | | |   update SCROLLBAR.LOWER_ALTITUDE to indicate new value
| | | | |
| | | | CASE SCROLLBAR.LOWER_ALTITUDE:
| | | | |   update FIELD_ENTRY.LOWER_ALTITUDE to indicate new value
| | | | |
| | | | CASE FIELD_ENTRY.UPPER_ALTITUDE:
| | | | |   USE f_cenh_validate_field(UPPER_ALTITUDE,&field_entry) TO perform range
| | | | |   check on field entry
| | | | |   update SCROLLBAR.UPPER_ALTITUDE to indicate new value
| | | | |
| | | | CASE SCROLLBAR.UPPER_ALTITUDE:
| | | | |   update FIELD_ENTRY.UPPER_ALTITUDE to indicate new value
| | | | |
| | | | CASE FIELD_ENTRY.WATER:
| | | | |   USE f_cenh_validate_field(WATER,&field_entry) TO perform range check on field
entry
| | | | |   update SCROLLBAR.WATER to indicate new value
| | | | |
| | | | CASE SCROLLBAR.WATER:
| | | | |   update FIELD_ENTRY.WATER to indicate new value
| | | | |
| | | | CASE FIELD_ENTRY.TEMPERATURE:
| | | | |   USE f_cenh_validate_field(TEMPERATURE,&field_entry) TO perform range
| | | | |   check on field entry
| | | | |   update SCROLLBAR.TEMPERATURE to indicate new value
| | | | |
| | | | CASE SCROLLBAR.TEMPERATURE:
| | | | |   update FIELD_ENTRY.TEMPERATURE to indicate new value
| | | | |
| | | | CASE FIELD_ENTRY.WIND_SPEED:
| | | | |   USE f_cenh_validate_field(SPEED,&field_entry) TO perform range check on field
entry
| | | | |   update SCROLLBAR.WIND_SPEED to indicate new value
| | | | |
| | | | CASE SCROLLBAR.WIND_SPEED:
| | | | |   update FIELD_ENTRY.WIND_SPEED to indicate new value
| | | | |
| | | | CASE FIELD_ENTRY.WIND_DIRECTION:
| | | | |   USE f_cenh_validate_field() TO perform range check on field entry
| | | | |   update SCROLLBAR.WIND_DIRECTION to indicate new value
| | | | |
| | | | CASE SCROLLBAR.WIND_DIRECTION:
| | | | |   update FIELD_ENTRY.WIND_DIRECTION to indicate new value
| | | | |
| | | | CASE PUSHBUTTON.OK
| | | | |   check that lower<upper
| | | | |   copy user data into database
| | | | |   IF (new cell was defined) THEN
| | | | |   |   USE f_cenh_edit_object(c_cell,&t_cend_env_def.cell[cell_count],
| | | | |   |   cell_count, c_add_element) TO add a precipitation cell to the CSS
| | | | |   ELSE
| | | | |   |   USE f_cenh_edit_object(c_cell,&t_cend_env_def.cell[cell_count],
| | | | |   |   cell_count, c_replace_element) TO edit a precipitation cell in the CSS
| | | | |   ENDIF
| | | | |   USE f_cenh_redraw() TO redraw the FOV display with the new data
| | | | |   USE UxPopdownInterface(Env_Cell_Iface) TO close the
| | | | |   precipitation cell definition data entry screen
| | | | |
| | | | CASE PUSHBUTTON.CANCEL
| | | | |   USE UxPopdownInterface(Env_Cell_Iface) TO close the
| | | | |   precipitation cell definition data entry screen
| | | | |
| | | | CASE PUSHBUTTON.APPLY
```

```
                    check that lower<upper
                    copy user data into database f_o_r current cell
                    IF (new cell was defined) THEN
                        USE f_cenh_edit_object(c_cell,&t_cend_env_def.cell[cell_count],
                        cell_count, c_add_element) TO add a precipitation cell to the CSS
                    ELSE
                        USE f_cenh_edit_object(c_cell,&t_cend_env_def.cell[cell_count],
                        cell_count, c_replace_element) TO edit a precipitation cell in the CSS
                    ENDIF
                    USE f_cenh_redraw() TO redraw the FOV display with the new data

            ENDSWITCH

            ENV_DISCRETE_CLUTTER:
            SWITCH Event

                CASE FIELD_ENTRY.DISCRETE_ID:
                    USE f_cenh_validate_field(DISCRETE_ID,&field_entry) TO perform range
                    check on field entry
                    update data fields with discrete clutter parameters f_o_r specified ID

                CASE FIELD_ENTRY.LAT:
                    USE f_cenh_validate_field(LATITUDE,&field_entry) TO perform range
                    check on field entry
                    update SCROLLBAR.LAT to indicate new value

                CASE SCROLLBAR.LAT:
                    update FIELD_ENTRY.LAT to indicate new value

                CASE FIELD_ENTRY.LONG:
                    USE f_cenh_validate_field(LONGITUDE,&field_entry) TO perform range
                    check on field entry
                    update SCROLLBAR.LONG to indicate new value

                CASE SCROLLBAR.LONG:
                    update FIELD_ENTRY.LONG to indicate new value

                CASE OPTION_MENU.CLUTTER_TYPE.RURAL_BUILDING:
                CASE OPTION_MENU.CLUTTER_TYPE.METAL_ROOF:
                CASE OPTION_MENU.CLUTTER_TYPE.MOBILE_HOME:
                    update field entries to appropriate values f_o_r clutter type

                CASE FIELD_ENTRY.MEDIAN:
                    USE f_cenh_validate_field(MEDIAN,&field_entry) TO perform range
                    check on field entry
                    update SCROLLBAR.MEDIAN to indicate new value

                CASE SCROLLBAR.MEDIAN:
                    update FIELD_ENTRY.MEDIAN to indicate new value

                CASE FIELD_ENTRY.PEAK:
                    USE f_cenh_validate_field(PEAK,&field_entry) TO perform range check on field entry
                    update SCROLLBAR.PEAK to indicate new value

                CASE SCROLLBAR.PEAK:
                    update FIELD_ENTRY.PEAK to indicate new value

                CASE FIELD_ENTRY.SPECTRAL:
                    USE f_cenh_validate_field(SPECTRAL,&field_entry) TO perform range
                    check on field entry
                    update SCROLLBAR.SPECTRAL to indicate new value

                CASE SCROLLBAR.SPECTRAL:
                    update FIELD_ENTRY.SPECTRAL to indicate new value

                CASE FIELD_ENTRY.DISCRETE_CLUTTER_SEED:
                    USE f_cenh_validate_field(SEED:,&field_entry) TO perform
                    range check on field entry

                CASE PUSHBUTTON.OK
                    copy user data into database
                    IF (new discrete clutter was defined) THEN
                        USE f_cenh_edit_object(c_discrete,&t_cend_env_def.discrete[discrete_count],
                        discrete_count, c_add_element) TO add discrete clutter to the CSS
                    ELSE
                        USE f_cenh_edit_object(c_discrete,&t_cend_env_def.discrete[discrete_count],
                        discrete_count, c_replace_element) TO edit discrete clutter in the CSS
                    ENDIF
                    USE f_cenh_redraw() TO redraw the FOV display with the new data
                    USE UxPopdownInterface(Env_Discrete_Clutter_Patch_Iface) TO close the
                    discrete clutter definition data entry screen

                CASE PUSHBUTTON.APPLY
```

```
    |   |   |   |   |   copy user data into database f_o_r current clutter patch
    |   |   |   |   |   IF (new discrete clutter was defined) THEN
    |   |   |   |   |   |   USE f_cenh_edit_object(c_discrete,&t_cend_env_def.discrete[discrete_count],
    |   |   |   |   |   |   discrete_count, c_add_element) TO add discrete clutter to the CSS
    |   |   |   |   |   ELSE
    |   |   |   |   |   |   USE f_cenh_edit_object(c_discrete,&t_cend_env_def.discrete[discrete_count],
    |   |   |   |   |   |   discrete_count, c_replace_element) TO edit discrete clutter in the CSS
    |   |   |   |   |   ENDIF
    |   |   |   |   |   USE f_cenh_redraw() TO redraw the FOV display with the new data
    |   |   |   |   |
    |   |   |   |   CASE PUSHBUTTON.CANCEL
    |   |   |   |   |   USE UxPopdownInterface(Env_Discrete_Clutter_Patch_Iface) TO close the
    |   |   |   |   |   discrete clutter definition data entry screen
    |   |   |   |   |
    |   |   |   ENDSWITCH
    |   |   |
    |   |   |   ENV_SEED:
    |   |   |   SWITCH Event
    |   |   |   |
    |   |   |   |   CASE FIELD_ENTRY.NOISE_SEED:
    |   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
    |   |   |   |   |
    |   |   |   |   CASE FIELD_ENTRY.IONOSPHERE_SEED
    |   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
    |   |   |   |   |
    |   |   |   |   CASE FIELD_ENTRY.CLUTTER_SEED:
    |   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
    |   |   |   |   |
    |   |   |   |   CASE FIELD_ENTRY.VOLUMETRIC_CLUTTER_SEED:
    |   |   |   |   |   USE f_cenh_validate_field() TO perform range check on field entry
    |   |   |   |   |
    |   |   |   |   CASE PUSHBUTTON.OK
    |   |   |   |   |   copy user data into database
    |   |   |   |   |   USE UxPopdownInterface(Env_Seed_Iface) TO close the
    |   |   |   |   |   seed definition data entry screen
    |   |   |   |   |
    |   |   |   |   CASE PUSHBUTTON.CANCEL
    |   |   |   |   |   USE UxPopdownInterface(Env_Seed_Iface) TO close the
    |   |   |   |   |   seed definition data entry screen
    |   |   |   |   |
    |   |   |   ENDSWITCH
    |   ENDSWITCH
    |
    ENDLOOP

*/
}

/* end of cenh_main.c */
```

```
/*%%===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:       cenh_redraw.c

Identification: [@(#)]

SCCS Id:         %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:        Rob Soossar

Change History:

SCR   Date        Author   Change made
---  ----------- ------  ---------------------------------------------


CSU_NAME:      HMI

FUNCTION:      f_cenh_redraw

PURPOSE:    This function is a Motif callback funtion to handle EXPOSE events
on the Environment HMI FOV display optimal handling of these
events will be determined at implementation.




INPUT                                  SOURCE
-----                                  ------

OUTPUT                                  DESTINATION
------                                  -----------


CREATION DATE:      June 03 1993

============================================================================*/
#include "cenh.h"

static void f_cenh_redraw(Widget w, caddr_t client_data, XEvent *event)
    /* Environment Definition HMI FOV Expose handler function */
{

/*|

    static XRectangle rect[c_max_redraw_rects];
    static int num_rect=0;
    Pescape_in_data esc_in;

    #if defined PUESC_REDRAW_REGIONS


    rect[num_rect].x = event->xexpose.x;
    rect[num_rect].y = event->xexpose.y;
    rect[num_rect].width = event->xexpose.width;
    rect[num_rect].height = event->xexpose.height;

    num_rect++;

    IF ( (event->xexpose.count == 0)|| (num_rects==MAX_REDRAW_RECTS)) THEN
    |   this is the last EXPOSE event to be handled
    |   esc_in.escape_in_u6.ws_id = WS_ID
    |   esc_in.escape_in_u6.num_regions = num_rects
    |   esc_in.escape_in_u6.regions = rect
    |   USE pescape(PUESC_REDRAW_REGIONS, &esc_in, (Pstore)NULL, (Pescape_out_data **)NULL)
    |   num_rects = 0
    ENDIF

    #else
    IF (event->xexpose.count == 0) THEN this is the last EXPOSE event to be handled
    |   USE predraw_all_structs(WS_ID, PFLAG_ALWAYS) to redraw the FOV
    ENDIF

    #endif
*/
}

/* end of cenh_redraw.c */
```

```
/*%%===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_save.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR    Date         Author    Change made
---  -----------  ------  ---------------------------------------------------


CSU_NAME:     HMI

FUNCTION:     f_cenh_save

PURPOSE:    This function will save Environment Definition data into the
specified file.




INPUT                                   SOURCE
-----                                   ------

OUTPUT                                   DESTINATION
------                                   -----------


CREATION DATE:      June 03 1993

===========================================================================*/
#include "cenh.h"

void f_cenh_save(char *file_name)
     /* Environment Definition HMI Task */
{
/*|


     USE fopen() TO open the Environment Definition data file

     IF error on open THEN
     |    report file open error
<--------return
     ELSE
     |    USE xdrstdio_create TO initialize a record oriented XDR stream
     |    object (xdrs) in XDR_DECODE mode
     |    USE xdr_t_write_hmi TO write Environment Definition data t_cend_env_def struct into file
     |    IF error on write THEN
     |    |    report file write error
<-----------return
     |    ENDIF
     |    USE xdr_destroy TO destroy xdrs
     ENDIF

     set data_status=UNCHANGED
*/
}

/* end of cenh_save.c */
```

```
/*%%===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_select_list.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR   Date        Author   Change made
---  -----------  ------  --------------------------------------------------


CSU_NAME:       HMI

FUNCTION:       f_cenh_select_list

PURPOSE:    This function will operate on the selection list with updates
of new items or highlighting a specified item.


INPUT                                    SOURCE
-----                                    ------

OUTPUT                                    DESTINATION
------                                    -----------


CREATION DATE:      June 03 1993

============================================================================*/
#include "cenh.h"

void f_cenh_select_list(int item_num, int edit_op, t_spd_location *loc)
    /* Environment Definition HMI Task */
{
/*|

    SWITCH edit_op
    |
    |   CASE c_add:
    |   |   add a new entry to the list in the appropriate location
    |   |
    |   CASE c_delete;
    |   |   remove an entry from the list
    |   |
    |   CASE c_highlight:
    |   |   change the color of an entry to indicate it is highlighted
    |   |
    ENDSWITCH
*/
}

/* end of cenh_select_list.c */
```

```
/*%%==========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the
Minister of National Defence, 1992

File Name:      cenh_valid.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Rob Soosaar

Change History:

SCR   Date        Author   Change made
---  -----------  ------  ---------------------------------------------


CSU_NAME:     HMI

FUNCTION:     f_cenh_validate_field

PURPOSE:    This function performs a field validation f_o_r the specified parameter.
A boolean value of TRUE is returned if the parameter is within range
bounds.


INPUT                                SOURCE
-----                                ------

OUTPUT                                DESTINATION
------                                -----------


CREATION DATE:     June 09 1993

===========================================================================*/
#include "cenh.h"

Boolean f_cenh_validate_field(char *field_entry_ptr, int field_type)
    /* Environment Def validion HMI Task */
{
/*|

    return_value = TRUE

    SWITCH field_type
    |
    |   CASE ELECTRON_DENSITY:
    |   |   USE sscanf() TO convert ascii data to binary format
    |   |   IF (input stream contains non-numeric or extraneous charachters) THEN
    |   |   |   return_value=FALSE
    |   |   ELSEIF (value outside of 10**9 to 10**14 m**(-3) range) THEN
    |   |   |   return_value=FALSE
    |   |   ENDIF
    |   |
    |   CASE ION_ALTITUDE:
    |   |   USE sscanf() TO convert ascii data to binary format
    |   |   IF (input stream contains non-numeric or extraneous charachters) THEN
    |   |   |   return_value=FALSE
    |   |   ELSEIF (value outside of 100 to 300 km. range) THEN
    |   |   |   return_value=FALSE
    |   |   ENDIF
    |   |
    |   CASE SCALE_HEIGHT:
    |   |   USE sscanf() TO convert ascii data to binary format
    |   |   IF (input stream contains non-numeric or extraneous charachters) THEN
    |   |   |   return_value=FALSE
    |   |   ELSEIF (value outside of 0 to 50 km. range) THEN
    |   |   |   return_value=FALSE
    |   |   ENDIF
    |   |
    |   CASE PATCH_ID:
    |   |   check if specified ID exists in database
    |   |   IF (ID exists) THEN
    |   |   |   return_value=TRUE
|    |   ELSE
|    |   |   return_value=FALSE
|    |   ENDIF
|    |
|    CASE SPEED:
|    |   USE sscanf() TO convert ascii data to binary format
|    |   IF (input stream contains non-numeric or extraneous charachters) THEN
|    |   |   return_value=FALSE
|    |   ELSEIF (value outside of 0 to 100 knots range) THEN
|    |   |   return_value=FALSE
|    |   ENDIF
|    |
|    CASE DIRECTION:
|    |   USE sscanf() TO convert ascii data to binary format
|    |   IF (input stream contains non-numeric or extraneous charachters) THEN
|    |   |   return_value=FALSE
|    |   ELSEIF (value outside of 0 to 360 degree range) THEN
|    |   |   return_value=FALSE
|    |   ENDIF
|    |
|    CASE TARGET_ID:
|    |   check if specified ID exists in database
|    |   IF (ID exists) THEN
|    |   |   return_value=TRUE
|    |   ELSE
|    |   |   return_value=FALSE
|    |   ENDIF
|    |
|    CASE LATITUDE:
|    |   USE sscanf() TO convert ascii data to binary format
|    |   IF (input stream contains non-numeric or extraneous charachters) THEN
|    |   |   return_value=FALSE
|    |   ELSEIF (value outside of 0 to 180 degree range) THEN
|    |   |   return_value=FALSE
|    |   ENDIF
|    |
|    CASE LONGITUDE:
|    |   USE sscanf() TO convert ascii data to binary format
|    |   IF (input stream contains non-numeric or extraneous charachters) THEN
|    |   |   return_value=FALSE
|    |   ELSEIF (value outside of 0 to 180 degree range) THEN
|    |   |   return_value=FALSE
|    |   ENDIF
|    |
|    CASE HEADING:
|    |   USE sscanf() TO convert ascii data to binary format
|    |   IF (input stream contains non-numeric or extraneous charachters) THEN
|    |   |   return_value=FALSE
|    |   ELSEIF (value outside of 0 to 360 degree range) THEN
|    |   |   return_value=FALSE
|    |   ENDIF
|    |
|    CASE TARGET_ALTITUDE:
|    |   USE sscanf() TO convert ascii data to binary format
|    |   IF (input stream contains non-numeric or extraneous charachters) THEN
|    |   |   return_value=FALSE
|    |   ELSEIF (value outside of 0 to 20000 meter range) THEN
|    |   |   return_value=FALSE
|    |   ENDIF
|    |
|    CASE TARGET_SPEED:
|    |   USE sscanf() TO convert ascii data to binary format
|    |   IF (input stream contains non-numeric or extraneous charachters) THEN
|    |   |   return_value=FALSE
|    |   ELSEIF (value outside of 0 to Mach 3 range) THEN
|    |   |   return_value=FALSE
|    |   ENDIF
|    |
|    CASE SPECTRAL_WIDTH:
|    |   USE sscanf() TO convert ascii data to binary format
|    |   IF (input stream contains non-numeric or extraneous charachters) THEN
|    |   |   return_value=FALSE
|    |   ELSEIF (value outside of 0 to 4 knots range) THEN
|    |   |   return_value=FALSE
|    |   ENDIF
|    |
|    CASE SEED:
|    |   USE sscanf() TO convert ascii data to binary format
|    |   IF (input stream contains non-numeric or extraneous charachters) THEN
|    |   |   return_value=FALSE
|    |   ELSEIF (value outside of 1 to 2^32-1 range) THEN
|    |   |   return_value=FALSE
|    |   ENDIF
|   |
|   CASE RCS:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of -10 to 40 dBm^2 range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE JAMMER_ID:
|   |   check if specified ID exists in database
|   |   IF (ID exists) THEN
|   |   |   return_value=TRUE
|   |   ELSE
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE JAMMER_SPEED:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 0 to 500 knots range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE ERP:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 50 to 120 dBW range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE START_TIME:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of CTI range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE BANDWIDTH:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 0 to 2 GHz range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE CENTRE_FREQUENCY:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 1 to 4 GHz range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE STROBE_FREQUENCY:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 0 to 100 kHz range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE DUTY_CYCLE:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 0 to 100 percent range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE SWEEP_FREQUENCY:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 1 to 4 GHz range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE PROCESSING_DELAY:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 0 to 10 msec range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE FALSE_TARGET:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 1 to 100  range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE CHAFF_BUNDLE_ID:
|   |   check if specified ID exists in database
|   |   IF (ID exists) THEN
|   |   |   return_value=TRUE
|   |   ELSE
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE WEIGHT:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 0 to 5 kg  range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE STAGGER_TUNE:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 1 to 500 MHz  range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE DISPERSION:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 0 to 100 percent range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE CELL_ID:
|   |   check if specified ID exists in database
|   |   IF (ID exists) THEN
|   |   |   return_value=TRUE
|   |   ELSE
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE CIRCULAR_RADIUS:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 0 to 60 km range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE LOWER_ALTITUDE:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (value outside of 0 to 8000 meter range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   CASE UPPER_ALTITUDE:
|   |   USE sscanf() TO convert ascii data to binary format
|   |   IF (input stream contains non-numeric or extraneous charachters) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (STATE is rain snow or cloud) AND (value outside of 0 to 8000 meter range) THEN
|   |   |   return_value=FALSE
|   |   ELSEIF (STATE is fog) AND (value outside of 0 to 100 meter range) THEN
|   |   |   return_value=FALSE
|   |   ENDIF
|   |
|   |
|   CASE WATER:
|   |   USE sscanf() TO convert ascii data to binary format
```

```
    |   |   IF (input stream contains non-numeric or extraneous charachters) THEN
    |   |   |   return_value=FALSE
    |   |   ELSEIF (STATE is rain ) AND (value outside of 0.25 to 40 mm/hr range) THEN
    |   |   |   return_value=FALSE
    |   |   ELSEIF (STATE is snow) AND (value outside of 0.025 to 4.0 mm/hr range) THEN
    |   |   |   return_value=FALSE
    |   |   ELSEIF (STATE is fog) AND (value outside of 0.04 to 2.3 gm/m^3 range) THEN
    |   |   |   return_value=FALSE
    |   |   ELSEIF (STATE is cloud) AND (value outside of 0.5 to 2.5 gm/m^3 range) THEN
    |   |   |   return_value=FALSE
    |   |   ENDIF
    |   |
    |   CASE TEMPERATURE:
    |   |   USE sscanf() TO convert ascii data to binary format
    |   |   IF (input stream contains non-numeric or extraneous charachters) THEN
    |   |   |   return_value=FALSE
    |   |   ELSEIF (value outside of -40 to +40 C range) THEN
    |   |   |   return_value=FALSE
    |   |   ENDIF
    |   |
    |   CASE DISCRETE_ID:
    |   |   check if specified ID exists in database
    |   |   IF (ID exists) THEN
    |   |   |   return_value=TRUE
    |   |   ELSE
    |   |   |   return_value=FALSE
    |   |   ENDIF
    |   |
    |   CASE MEDIAN:
    |   |   USE sscanf() TO convert ascii data to binary format
    |   |   IF (input stream contains non-numeric or extraneous charachters) THEN
    |   |   |   return_value=FALSE
    |   |   ELSEIF (value outside of 20 to 60 dBm^2 range) THEN
    |   |   |   return_value=FALSE
    |   |   ENDIF
    |   |
    |   CASE PEAK:
    |   |   USE sscanf() TO convert ascii data to binary format
    |   |   IF (input stream contains non-numeric or extraneous charachters) THEN
    |   |   |   return_value=FALSE
    |   |   ELSEIF (value outside of 20 to 70 dBm^2 range) THEN
    |   |   |   return_value=FALSE
    |   |   ENDIF
    |   |
    |   CASE SPECTRAL:
    |   |   USE sscanf() TO convert ascii data to binary format
    |   |   IF (input stream contains non-numeric or extraneous charachters) THEN
    |   |   |   return_value=FALSE
    |   |   ELSEIF (value outside of 0 to 1 knot range) THEN
    |   |   |   return_value=FALSE
    |   |   ENDIF
    |   |
    ENDSWITCH
*/
}

/* end of cenh_valid.c */
```

```
==================================================================================================================
==============================
PAGE NO.   LINE NO.          ENCLOSING FUNCTION

  NO TBDs TO REFERENCE
```

```
===========================================================================================================
=============================
PAGE NO.   LINE NO.


    14        905        f_cenh_init
    34       2456        |    f_cenh_redraw
     1         47        |    f_cenh_close
    36       2548        |    |    f_cenh_save

    20       1320        f_cenh_main
    17       1105        |    f_cenh_load
    16       1032        |    |    f_cenh_empty_fov
    37       2626        |    |    f_cenh_select_list
     2        128        |    |    f_cenh_edit_object
     2        128        |    |    |    f_cenh_edit_object   *** RECURSIVE ***
    34       2456        |    |    f_cenh_redraw
    36       2548        |    f_cenh_save
    19       1241        |    f_cenh_1d_radar
     1         47        |    f_cenh_close
    36       2548        |    |    f_cenh_save
    34       2456        |    f_cenh_redraw
    37       2626        |    f_cenh_select_list
     2        128        |    f_cenh_edit_object
     2        128        |    |    f_cenh_edit_object   *** RECURSIVE ***
    38       2699        |    f_cenh_validate_field

     5        349        f_cenh_handle_lmouse
     2        128        |    f_cenh_edit_object
     2        128        |    |    f_cenh_edit_object   *** RECURSIVE ***
    34       2456        |    f_cenh_redraw
    37       2626        |    f_cenh_select_list

     9        607        f_cenh_handle_mmouse
     2        128        |    f_cenh_edit_object
     2        128        |    |    f_cenh_edit_object   *** RECURSIVE ***
    34       2456        |    f_cenh_redraw

    12        813        f_cenh_handle_rmouse
     2        128        |    f_cenh_edit_object
     2        128        |    |    f_cenh_edit_object   *** RECURSIVE ***
```

```
              PAGE 45
                                                          ERROR SUMMARY

==================================================================================================================
==============================
PAGE NO.   LINE NO.    ERROR TEXT

   NO ERRORS TO REPORT
```

D1.3 Simulation Run HMI (CSDH)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csdh_priv.h - Simulation Run Definition HMI
                              private include file

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file declares constants (defaults and ranges) for
                a Satellite/Radar Definition.

Designer:       Darren McHugh

Change History:

    SCR   Date         Author   Change made
    ---   -----------  ------   -----------------------------------------
    n/a   26-Apr-1993  mchugh   Initial Release

============================================================================*/

/* RUN HMI 1.2.1 */
#define c_csdh_dfl_DPCA_mode_RRA          c_spd_DPCA_none
#define c_csdh_dfl_DPCA_mode_CFA          c_spd_ASAR_DPCA

/* RUN HMI 1.2.2 */
#define c_csdh_dfl_monopulse_enable          c_spd_false

/* RUN HMI 1.2.3 */
#define c_csdh_dfl_num_rg_dop_blankers                0
#define c_csdh_min_num_rg_dop_blankers_monopulse      0
#define c_csdh_max_num_rg_dop_blankers_monopulse      0
#define c_csdh_min_num_rg_dop_blankers_ASAR           0
#define c_csdh_max_num_rg_dop_blankers_ASAR           1
#define c_csdh_min_num_rg_dop_blankers_non_ASAR       0
#define c_csdh_max_num_rg_dop_blankers_non_ASAR       2

/* RUN HMI 2.1.2.1 */
#define c_csdh_dfl_num_MASR_ph_ctr           3
#define c_csdh_min_num_MASR_ph_ctr           2
#define c_csdh_max_num_MASR_ph_ctr           3

/* RUN HMI 2.1.2.2 */
#define c_csdh_dfl_monopulse_lobe_sel    c_spd_across_groups

/* RUN HMI 2.1.2.3 */
#define c_csdh_dfl_DPCA_group_sel            1
#define c_csdh_min_DPCA_group_sel            1
#define c_csdh_max_DPCA_group_sel            2

/* RUN HMI 2.1.2.4 */
#define c_csdh_dfl_ph_ctr_pair_sel           c_spd_leading

/* RUN HMI 2.1.4 */
#define c_csdh_min_num_pre_dop_blankers      0
#define c_csdh_max_num_pre_dop_blankers      4

/* RUN HMI 2.2.1.1 */
#define c_csdh_dfl_snapshot_smooth_period_RRA   0.5  /* milliseconds */
#define c_csdh_dfl_snapshot_smooth_period_CFA   5   /* milliseconds */
```

```
#define c_csdh_min_snapshot_smooth_period       0.1   /* milliseconds */
#define c_csdh_max_snapshot_smooth_period      10.0   /* milliseconds */

/* RUN HMI 2.2.1.2 */
#define c_csdh_dfl_num_snapshots_RRA        30
#define c_csdh_dfl_num_snapshots_CFA        65
#define c_csdh_min_num_snapshots             1
#define c_csdh_max_num_snapshots           100

/* RUN HMI 2.2.1.3 */
#define c_csdh_dfl_num_update_intervals      1
#define c_csdh_min_num_update_intervals      1
#define c_csdh_max_num_update_intervals     99

/* RUN HMI 2.2.1.4 */
#define c_csdh_dfl_snapshot_selection       c_spd_smooth_uniform

/* RUN HMI 2.2.1.5 */
#define c_csdh_dfl_snapshot_edit            c_spd_none

/* RUN HMI 2.2.1.5.1 */
#define c_csdh_dfl_num_snapshots_rejected   1
#define c_csdh_min_num_snapshots_rejected   1
#define c_csdh_max_num_snapshots_rejected   99

/* RUN HMI 2.2.1.6 */
#define c_csdh_dfl_notch_filter_len         3
#define c_csdh_min_notch_filter_len         1
#define c_csdh_max_notch_filter_len         7

/* RUN HMI 2.2.1.6.1 */
#define c_csdh_dfl_notch_filter_type        c_spd_binomial_MTI

/* RUN HMI 2.2.1.7 */
#define c_csdh_dfl_diag_loading_const       0.0
#define c_csdh_min_diag_loading_const       0.0
#define c_csdh_max_diag_loading_const       1.0e-3

/* RUN HMI 2.2.1.8 */
#define c_csdh_dfl_stability_thrshd         1.0e-8
#define c_csdh_min_stability_thrshd         0.0
#define c_csdh_max_stability_thrshd         1.0e-3

/* RUN HMI 2.2.1.9 */
#define c_csdh_dfl_weight_size_thrshd_enable c_spd_false
#define c_csdh_min_weight_size_thrshd         1.0
#define c_csdh_max_weight_size_thrshd         1.0e4

/* RUN HMI 2.2.2.1 */
#define c_csdh_dfl_predop_blk_noise_thrshd   100.0
#define c_csdh_min_predop_blk_noise_thrshd   0.0
#define c_csdh_max_predop_blk_noise_thrshd   1.0e4

/* RUN HMI 2.2.2.2 */
#define c_csdh_dfl_predop_blk_slope_thrshd   0.03
#define c_csdh_min_predop_blk_slope_thrshd   0.0
#define c_csdh_max_predop_blk_slope_thrshd   1.0

/* RUN HMI 2.2.2.3 */
#define c_csdh_dfl_predop_blk_channel_comb   c_spd_channel_max

/* RUN HMI 2.2.3   */
#define c_csdh_dfl_MASR_enable               c_spd_true

/* RUN HMI 2.2.3.1 */
#define c_csdh_dfl_MASR_mode                 c_spd_adaptive_weights

/* RUN HMI 2.2.4.1 */
#define c_csdh_dfl_SC_enable_adj             c_spd_false
#define c_csdh_dfl_SC_enable_not_adj         c_spd_true
```

```

/* RUN HMI 2.2.4.1.1 */
#define c_csdh_dfl_SC_filter_len              4
#define c_csdh_min_SC_filter_len              4
#define c_csdh_max_SC_filter_len              32

/* RUN HMI 2.2.4.1.2 */
#define c_csdh_dfl_SC_filter_update_int       1
#define c_csdh_min_SC_filter_update_int       1
#define c_csdh_max_SC_filter_update_int       10000

/* RUN HMI 2.2.4.2 */
#define c_csdh_dfl_freq_align_enable          c_spd_true

/* RUN HMI 2.2.4.3 */
#define c_csdh_dfl_num_pulses_proc_RRA        74
#define c_csdh_dfl_num_pulses_proc_CFA        304
#define c_csdh_min_num_pulses_proc            32
#define c_csdh_max_num_pulses_proc            1024

/* RUN HMI 2.2.5 */
#define c_csdh_dfl_PD_enable                  c_spd_true

/* RUN HMI 2.2.5.1 */
#define c_csdh_dfl_PD_FFT_len_RRA             128
#define c_csdh_dfl_PD_FFT_len_CFA             512
#define c_csdh_min_PD_FFT_len                 32
#define c_csdh_max_PD_FFT_len                 1024

/* RUN HMI 2.2.5.2 */
#define c_csdh_dfl_PD_window_type             c_spd_hamming_window

/* RUN HMI 2.2.5.2.1 */
#define c_csdh_dfl_PD_kaiser_SL_lvl           -50.0   /* dB */
#define c_csdh_min_PD_kaiser_SL_lvl           -13.0   /* dB */
#define c_csdh_max_PD_kaiser_SL_lvl           -100.0  /* dB */

/* RUN HMI 2.2.5.3 */
#define c_csdh_dfl_doppler_aug_sel            c_spd_NCI_augment

/* Unambiguous PRF                            RUN HMI 2.2.5.3.1 */
#define c_csdh_dfl_unambig_PRF_RRA            20000   /* Hz */
#define c_csdh_dfl_unambig_PRF_CFA            10000   /* Hz */
#define c_csdh_min_unambig_PRF                100     /* Hz */
#define c_csdh_max_unambig_PRF                200000  /* Hz */

/* FIXED DEFAULT AND RANGE FOR PULSE COMPRESSION */
/* RUN HMI 2.2.6 */
#define c_csdh_dfl_pulse_compr_enable         c_spd_true

/* Doppler Compensated Replicas               RUN HMI 2.2.6.1 */
#define c_csdh_dfl_doppler_comp_enable        c_spd_true

/* Pulse compression Window Type              RUN HMI 2.2.6.2 */
#define c_csdh_dfl_PC_window_type             c_spd_hamming_window

/* Kaiser Sidelobe level                      RUN HMI 2.2.6.2.1 */
#define c_csdh_dfl_PC_kaiser_SL_lvl           -50.0     /* dB */
#define c_csdh_min_PC_kaiser_SL_lvl           -13.0     /* dB */
#define c_csdh_max_PC_kaiser_SL_lvl           -100.0    /* dB */

/* FIXED DEFAULT AND RANGE FOR ASAR DPCA          */
/* ASAR DPCA enable                           RUN HMI 2.2.7 */
#define c_csdh_dfl_ASAR_DPCA_enable           c_spd_true

/* Number of doppler bins                     RUN HMI 2.2.7.1 */
#define c_csdh_dfl_num_DPCA_cells             127
#define c_csdh_min_num_DPCA_cells             1
#define c_csdh_max_num_DPCA_cells             1023

```

```
/* Number of Range Gates Used to compute Weights   RUN HMI 2.2.7.2 */
#define c_csdh_dfl_num_cc_rng_gates          20
#define c_csdh_min_num_cc_rng_gates          1
#define c_csdh_max_num_cc_rng_gates          512

/* Range Gate Selection                      RUN HMI 2.2.7.3 */
#define c_csdh_dfl_rng_gate_type             c_spd_uniform_gate

/* Start of Range Gate                       RUN HMI 2.2.7.3.1 */
#define c_csdh_dfl_start_rng_gate            1
#define c_csdh_min_start_rng_gate            1
#define c_csdh_max_start_rng_gate            512
/* Step Range Gate                                             */
#define c_csdh_dfl_step_rng_gate             1
#define c_csdh_min_step_rng_gate             1
#define c_csdh_max_step_rng_gate             511

/* Target/clutter ratio to compute deweighting threshold  RUN HMI 2.2.7.4 */
#define c_csdh_dfl_deweight_TIR              -25.0      /* dB */
#define c_csdh_min_deweight_TIR              -100.0     /* dB */
#define c_csdh_max_deweight_TIR               0.0       /* dB */

/* FIXED RANGE AND DEFAULT PARAMETERS FOR RANGE DOPPLER BLANKING */

/* Blanking noise threshold                  RUN HMI 2.2.8.1 */
#define c_csdh_dfl_rgdop_blk_noise_thrshd    100
#define c_csdh_min_rgdop_blk_noise_thrshd    1
#define c_csdh_max_rgdop_blk_noise_thrshd    10000

/* Blanking Slope Threshold                  RUN HMI 2.2.8.2 */
#define c_csdh_dfl_rgdop_blk_slope_thrshd    0.03
#define c_csdh_min_rgdop_blk_slope_thrshd    0.0
#define c_csdh_max_rgdop_blk_slope_thrshd    1.0

/* Channel combination (maximum or sum)      RUN HMI 2.2.8.3 */
#define c_csdh_dfl_rgdop_blk_channel_comb    c_spd_channel_max

/* FIXED RANGE AND DEFAULT PARAMETERS FOR NCI */

/* number of bursts integrated               RUN HMI 2.2.9.1 */
#define c_csdh_dfl_num_bursts_integ_RRA      4
#define c_csdh_dfl_num_bursts_integ_CFA      6
#define c_csdh_min_num_bursts_integ          1
#define c_csdh_max_num_bursts_integ          64

/* Integration algorithm                     RUN HMI 2.2.9.2 */
#define c_csdh_dfl_integ_method              c_spd_linear

/* OS NCI rank                               RUN HMI 2.2.9.2.1 */
/* cannot define a static default for this parameter, as
it is dependent on other parameters                      */
#define c_csdh_min_OS_NCI_rank               1
#define c_csdh_max_OS_NCI_rank               64

/* target range walk compensation enable flag RUN HMI 2.2.9.3  */
#define c_csdh_dfl_target_rng_walk_comp_enable c_spd_true

/* sampling alignment enable flag            RUN HMI 2.2.9.4 */
#define c_csdh_dfl_sampl_align_enable        c_spd_true

/* velocity alignment enable flag            RUN HMI 2.2.9.4 */
#define c_csdh_dfl_vel_align_enable          c_spd_true

/* Nominal carrier frequency for velocity align. RUN HMI 2.2.9.5.1 */
#define c_csdh_dfl_nominal_carrier_freq      2.0e9  /* Hz */
#define c_csdh_min_nominal_carrier_freq_L    1.0e9  /* Hz */
#define c_csdh_max_nominal_carrier_freq_L    2.0e9  /* Hz */
#define c_csdh_min_nominal_carrier_freq_S    2.0e9  /* Hz */
#define c_csdh_max_nominal_carrier_freq_S    4.0e9  /* Hz */

```

```
/* Nominal PRF for vel align.               RUN HMI 2.2.9.5.2 */
#define c_csdh_dfl_nominal_PRF              10000   /* Hz */
#define c_csdh_min_nominal_PRF              100     /* Hz */
#define c_csdh_max_nominal_PRF              200000  /* Hz */

/* FIXED RANGE AND DEFAULT PARAMETERS FOR DETECTION (CFAR) */

/* Type of threshold calculation used       RUN HMI 2.2.10.1 */
#define c_csdh_dfl_detect_type              c_spd_CA_CFAR

/* CFAR Reference Cells                     RUN HMI 2.2.10.2 */
#define c_csdh_dfl_doppler_width_RRA        1
#define c_csdh_dfl_doppler_width_CFA        35
#define c_csdh_min_doppler_width            1
#define c_csdh_max_doppler_width            2048

#define c_csdh_dfl_range_width_RRA          35
#define c_csdh_dfl_range_width_CFA          1
#define c_csdh_min_range_width              1
#define c_csdh_max_range_width              512

#define c_csdh_dfl_guard_doppler_width_RRA  1
#define c_csdh_dfl_guard_doppler_width_CFA  3
#define c_csdh_min_guard_doppler_width      0
#define c_csdh_max_guard_doppler_width      2048

#define c_csdh_dfl_guard_range_width_RRA    3
#define c_csdh_dfl_guard_range_width_CFA    1
#define c_csdh_min_guard_range_width        0
#define c_csdh_max_guard_range_width        512

/* OS rank for CFAR type                    RUN HMI 2.2.10.3 */
#define c_csdh_dfl_OS_CFAR_rank_RRA         24
#define c_csdh_dfl_OS_CFAR_rank_CFA         24
#define c_csdh_min_OS_CFAR_rank             1
#define c_csdh_max_OS_CFAR_rank             1e6

/* Probability of false alarm               RUN HMI 2.2.10.4 */
#define c_csdh_dfl_prob_false_alarm         1.000e-6
#define c_csdh_min_prob_false_alarm         1.0e-9
#define c_csdh_max_prob_false_alarm         1.0e-1

/* FIXED RANGE AND DEFAULT PARAMETERS FOR RAR */
/* Number of PRFs for range ambiguity resolution RUN HMI 2.2.11.1 */
#define c_csdh_dfl_num_PRF_used             3
#define c_csdh_min_num_PRF_used             1
#define c_csdh_max_num_PRF_used             12

/* footprint length                         RUN HMI 2.2.11.2 */
#define c_csdh_dfl_footprint_len            1000   /* km */
#define c_csdh_min_footprint_len            1      /* km */
#define c_csdh_max_footprint_len            10000  /* km */


/* ------------------------------------------------------------
            DATA CAPTURE OPTIONS
   ------------------------------------------------------------ */

/* FIXED RANGE AND DEFAULT PARAMETERS FOR ESP SIGNAL DATA */
/* Options for each channel                 RUN HMI 2.3.1 */
#define c_csdh_dfl_capt_ESP_input           c_spd_false
#define c_csdh_dfl_capt_post_SC             c_spd_false
#define c_csdh_dfl_capt_nulled              c_spd_false
#define c_csdh_dfl_capt_ESP_blanked         c_spd_false
#define c_csdh_dfl_capt_post_MASR           c_spd_false
#define c_csdh_dfl_capt_post_FA             c_spd_false
#define c_csdh_dfl_capt_post_PD             c_spd_false


/* FIXED RANGE AND DEFAULT PARAMETERS FOR RSP SIGNAL DATA */
/* Options for each channel                 RUN HMI 2.3.2 */
#define c_csdh_dfl_capt_RSP_input           c_spd_false
#define c_csdh_dfl_capt_post_PC             c_spd_false
#define c_csdh_dfl_capt_post_CC             c_spd_false
#define c_csdh_dfl_capt_monopulse_angle     c_spd_false
#define c_csdh_dfl_capt_RCS                 c_spd_false
#define c_csdh_dfl_capt_RSP_blanked         c_spd_false
#define c_csdh_dfl_capt_post_NCI            c_spd_false

/* FIXED RANGE AND DEFAULT PARAMETERS FOR SIGNAL DATA REDUCTION */
/* Options for each channel                 RUN HMI 2.3.3 */
#define c_csdh_dfl_time_decim_start         1
#define c_csdh_min_time_decim_start         1
#define c_csdh_max_time_decim_start         1024
#define c_csdh_dfl_time_decim_stop          64
#define c_csdh_min_time_decim_stop          1
#define c_csdh_max_time_decim_stop          1024
#define c_csdh_dfl_time_decim_step          1
#define c_csdh_min_time_decim_step          1
#define c_csdh_max_time_decim_step          1023

#define c_csdh_dfl_range_decim_start         1
#define c_csdh_min_range_decim_start         1
#define c_csdh_max_range_decim_start         512
#define c_csdh_dfl_range_decim_stop          64
#define c_csdh_min_range_decim_stop          1
#define c_csdh_max_range_decim_stop          512
#define c_csdh_dfl_range_decim_step          1
#define c_csdh_min_range_decim_step          1
#define c_csdh_max_range_decim_step          511

#define c_csdh_dfl_doppler_decim_start          1
#define c_csdh_min_doppler_decim_start          1
#define c_csdh_max_doppler_decim_start          1024
#define c_csdh_dfl_doppler_decim_stop           64
#define c_csdh_min_doppler_decim_stop           1
#define c_csdh_max_doppler_decim_stop           1024
#define c_csdh_dfl_doppler_decim_step           1
#define c_csdh_min_doppler_decim_step           1
#define c_csdh_max_doppler_decim_step           1023

#define c_csdh_dfl_aug_doppler_decim_start          1
#define c_csdh_min_aug_doppler_decim_start          1
#define c_csdh_max_aug_doppler_decim_start          2048
#define c_csdh_dfl_aug_doppler_decim_stop           64
#define c_csdh_min_aug_doppler_decim_stop           1
#define c_csdh_max_aug_doppler_decim_stop           2048
#define c_csdh_dfl_aug_doppler_decim_step           1
#define c_csdh_min_aug_doppler_decim_step           1
#define c_csdh_max_aug_doppler_decim_step           2047

#define c_csdh_dfl_magnitude_sqrd                   c_spd_false
#define c_csdh_dfl_capture_CPI_enable               c_spd_true
#define c_csdh_dfl_capture_DI_enable                c_spd_true
#define c_csdh_dfl_capture_NCI_enable               c_spd_true
#define c_csdh_dfl_capture_CPI_enable               c_spd_true

/* FIXED RANGE AND DEFAULT PARAMETERS FOR NORMAL EQUATIONS PRODUCTS */
/* RUN HMI 2.3.4 */
#define c_csdh_dfl_capt_weight_vectors      c_spd_false

/* Capture R-Matrices and r-vectors  */
#define c_csdh_dfl_capt_R_r                 c_spd_false

/* Capture U-Matrices and z-vectors  */
#define c_csdh_dfl_capt_U_z                 c_spd_false

/* Capture snapshots                 */
#define c_csdh_dfl_capt_snapshots           c_spd_false

```

```
/* FIXED RANGE AND DEFAULT PARAMETERS FOR DSP PRODUCTS */
/* RUN HMI 2.3.5 */
/* capture pulse doppler window */
#define c_csdh_dfl_capt_PD_window            c_spd_false

#define c_csdh_dfl_capt_DPCA_weights         c_spd_false

/* capture target deweighting function */
#define c_csdh_dfl_capt_target_deweight      c_spd_false

#define c_csdh_dfl_capt_PC_replicas          c_spd_false


```

```

#define c_crad_max_doppler_decim_start  2048    /* integer */
#define c_crad_max_doppler_decim_stop   2048    /* integer */
#define c_crad_max_doppler_decim_step   2047    /* integer *

```

```
/*============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csdh_main.c - Main program of Controller Simulation Run
                              Definition HMI task.

Identification: [@(#)]

SCCS Id:        @(#)csdh_main.c 1.1 93/04/22

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file implements the main program for the
                Simulation Run Definition HMI task.


Designer:       Stephen Tran

Change History:

   SCR  Date        Author   Change made
   ---  ----------- ------  ---------------------------------------------
   n/a 29-Jan-1993  cst     Initial Release

============================================================================*/

/* Included external declarations */
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include "spd_intf.h"
#include "csid_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* common constants */
/* none */

/* Implementation */
/*============================================================================

   program:     csdh_main

   Description:
      Main program to run Controller Simulation Run Definition HMI task.

      This is a stand-alone task which provides capability to define
      a Satellite Run Definition through Motif widgets and dialogues
      in X window environment.

      This task is implemented using UIM/X tool with X and Motif widgets.
      UIM/X is a user interface management tool; all code can be entered
      without leaving the application itself and UIM/X provides a text
      editor function for editing C code. All widget resources (core,
      specific, behavior, declaration, constraint) defined for an interface
      are kept in an interface file (.i); one file per interface.

      These interface files are created automatically by UIM/X. The C code
      is created either calling uxcgen utility from the UNIX command line
```

```
or pressing the "Write C code" button in UIM/X. Typically, the
interface files are submitted for Configuration Management and
the C code is created for compiling and linking via makefile.

In essence, most of the CSDH CSU codes are implemented in X callback
functions (how the widget reacts to events). The UIM/X tool supports
the generation of the callback function as well as the widget
creation and display function in C code; one .C file per interface.
We have to compile the generated C code and link them into application
executable.

This task will have a main program, which calls various initialization
routines and then enters XtAppMainLoop, awaiting x events and never
to return.  The user can terminate this task by selecting the "EXIT"
menu.

A Simulation Run top-level visualization window (a control pannel
window) will be displayed. This visualization window shows the
Simulation Run parameter chain and provides capability to
define a definition by applying the edit operation to parameter groups.

Please note that reading of the header of a RSS derived processing
parameters file is necessary to validate the consistency between
the simulated radar signal file (output from generation activity and
is used for data playback by the RSS) and the current simulation run
definition.

If the user selects "OPEN" menu, a file list is displayed for
definition selection.  After an existing file is selected, the
definition file is read into in-memory definition structure.
On the other hand, if the user selects "NEW" menu, the in-memory
structure will be initialized with default parameter values.
The definition structure is defined in the package CSID.

Other low-level dialog windows will be displayed as a result of
some widget selection in the top-level visualization window.
Each parameter definition will be implemented in a form of a MOTIF
widget (push button, option button, scaler, text, etc.)
A callback function is invoked every time a parameter value is
modified.  Note that defaults are always provided for a new definition
or for a new interval (DI, NCI, CPI).

Callback functions extract parameter values from a in-memory structure
for display as well as extract parameter values from the display
and place them back into the in-memory definition structure when
the user hits the "OK" button in the dialog window. No update is
performed in the in-memory structure, if the user presses the "CANCEL"
button.

Callback functions also perform range validation, implements
structural dependency logic as well as some parameter dependency
logic by ghosting the widgets not associated with some previously
entered data. See Simulation Run HMI table.

Most of range validation and parameter dependency checking
will be performed using functions provided by package CVAL.
Validation will be invoked, every time when the user presses
the "VALIDATE" menu button. The validation message log will
be displayed and the user can optionally save the validation log
to a user-specified file.

This task also performs consistency checking of the defined Simulation
Run with a simulated radar signal data, output of the previous signal
generation.  Because the header of RSS derived processing parameters
file contains a subset of Satellite/Radar Definition parameters,
the user can select which files he wants to validate the Simulation
Run Definition to be consistent with.  The RSS derived processing
parameters file is selected through a file selection list.

As defined in package CGEP, the header block contains satellite
```

```
   configuration (concepts, the number of auxiliaries, delay taps
   and phase centres) as well as the number of dwells, NCIs and CPIs,
   and sampling rate etc.  These parameters are used to check against
   the ESP and RSP configuration and other processing parameters
   in the Simulation Run Definition in the "VALIDATE" function.
   See Simulation Run HMI table.

   After the definition is completed, the user can select "SAVE" or
   "SAVE-AS" menu to write out the in-memory defintion structure
   into a UNIX external data file in XDR format.  Note that
   the definition structure is defined in the CRAD package and
   the XDR routines for external data convertion are provided by
   the same package.

   Command:  csdh_main 2>log_pathname &

   This task will log unsolicited errors occured in X/Motif calls
   to standard error and the error messages will then be
   re-directed to the user-specified pathname.

Globle variables:
   g_sim_def         - in-memory Simulation Run Definition data structure
   g_gen_header      - in-memory Generation Header (from RSS derived
                       processing parameters file) which contains a subset
                       of Satellite/Radar Definition

Exceptions:
   Errors logged by RSS and DSP and any errors occurred in the Controller
   Simulation Server for a simulation activity will be merged into
   one error file with prefix of run_ident.

Notes:
   This file is for the main program only, all dialog creation
   and callback functions will be included in different files
   (generated by UIM/X) in this package; one file per interface.

High Level PDL

   BEGIN csdh_main
      USE f_CERR_initialize() TO initialize log file
      get node names of Controller and RSS from environment variables:
        CTLR_NODE and RSS_NODE
      get default directory name for Simulation Run Definition file
      USE XtAppInitialize TO initialize the X Application
      USE UIM/X routines TO create required widgets for the top-level
          visualization window
      USE XtAppMainLoop() TO enter infinite loop and dispatch X event
        to the appropriate callback functions
   END csdh_main

   BEGIN user interface services functions
      { these functions will be implemented in different callback functions }

      case user presses "SELECT GENERATION" :
         user selects RSS derived processing parameters pathname
           (directory + filename) through a selection list
         read header from RSS derived processing parameters into g_gen_header

      case user presses "OPEN" :
         user selects Simulation Run Definition pathname (directory +
           filename) through a selection list
         USE f_csid_read TO read the specified data file into
           in-memory definition data structure

      case user presses "NEW" :
         initialize the in-memory definition data structure

      case user presses "SAVE" :
         USE f_csid_write TO write out the in-memory data structure to
           external data file

         case user presses "SAVE-AS" :
            user enters directory name and file name for the Simulation Run
              Definition data file
            USE f_csid_write TO write out the in-memory data structure to
              external data file

         case user presses "VALIDATE" :
            USE f_cval_sim_run_def TO validate the in-memory simulation run
               data structure (g_sim_def) and the consistency between
               the simulation run definition and the header of RSS derived
               processing parameters file (g_gen_header)
            IF error THEN
               display the validation log to the error message area
            ENDIF

      END user interface service functions
   End of PDL
==========================================================================*/

/* Routine Declaration */
t_spd_status main (
      t_spd_int32       argc,
      char              **argv,
      char              **envp)
{   /* Begin of main */
}   /* End of main */
```

D1.4 RSS Generation Interface (CRGI)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*====================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      crgi_intf.x - Controller/RSS Radar Signal Generation Interface

                CRGI is a package under the RIF [RSS InterFace]
                CSC in the Controller functional area. CRGI Specifies the
                Controller/RSS interface for Signal Generation
                Activities only.

Identification: [@(#)]

SCCS Id:        @(#)crgi_intf.x 1.5 93/06/25

Responsible Organization:

                MacDonald Dettwiler & Associates

Purpose:        This package declares RPC interfaces (constants and message
                types) between the Generation Server task in Controller
                functional area and the RSS Generation Manager task in the
                RSS functional area for a signal generation activity.
                The message definitions and protocol are based on the
                Testbed Preliminary Design Document.

                This file is written in RPC data description language in XDR
                (eXternal Data Representation) standard. This file will use
                "rpcgen" utility to generate the following two files:

                crgi_intf.h     - global include file for both Controller
                                  and RSS generation task packages to include;
                crgi_intf_xdr.c - XDR source file which contains required XDR
                                  conversion routines to support the RPC
                                  message data types.

                The generated files are needed for building the executables for
                both Testbed Generation Server and RSS Generation Manager Task.

Designer:       Rob Gottschalk

Change History:

    SCR   Date          Author   Change made
    ---   -----------   ------   ----------------------------------------
    n/a   06-Jan-1993   robg     Initial Release
          04-Feb-1993   cst      overall update; add protocol description
          09-Mar-1993   cst      after 40% review
                                 change c_crgi_completed_normally to
                                        c_crgi_gen_compl
                                        c_crgi_terminated_with_error to
                                        c_crgi_gen_failed
                                        c_crgi_abort_completed to
                                        c_crgi_abort_compl
                                        *gen_command to *gencmd
                                        *gen_rsp to *genrsp
                                 add c_crgi_ping_cmd and c_crgi_status_cmd
          03-May-1993   cst      added c_crgi_sgen_lognam (filename of RSS
                                 generation error log)
          22-Jun-1993   cst      replaced c_crgi_sgen_lognam by
                                   c_crgi_log_file to include directory name
                                   (Design Issue #28)
                                 replaced output_dir by SRSD_dir and prod_dir


 ====================================================================*/

```

```
%/*
% *
% *      PROCESS STARTUP/SHUTDOWN
% *
% *      RSS Signal Generation Manager works as a server (daemon process)
% *      for Controller Generation Task, while the Controller Generation
% *      Task is a server (daemon process) serving Generation Run HMI tasks.
% *      Both Controller Generation Server and the RSS Signal Generation Manager
% *      will be running all the time and await for generation requests either
% *      from the operator and from a batch program.
% *
% *      Both Controller Generation Server and RSS Generation Manager will
% *      be launched or shutdown by the SPD System Manager through a script
% *      file in which an offline program will be invoked. (See package CSYS).
% *
% *      When the RSS Generation Manager receives the shutdown request,
% *      RSS should gracefully bring down all related child and daemon
% *      processes for signal generation activity in RSS and then terminate
% *      the RSS Generation Manager itself.
% *
% *      MESSAGE TYPES:
% *
% *      1) RPC Command messages: sent from the Controller to RSS
% *
% *         Asynchronous message:
% *
% *         a) start generation command   - to start a signal generation activity
% *         b) abort generation command   - to stop the current generation
% *                                         activity
% *
% *         Synchronous message:
% *
% *         c) shutdown generation command - to shutdown RSS signal generation
% *                                          processes
% *         d) provide status message     - request the current generation status
% *                                         (running or idle)
% *         e) ping message               - request an echo-response
% *
% *      2) RPC Response messages: sent from RSS to the Controller
% *
% *         Asynchronous message:
% *
% *         a) start generation response - to respond to the start generation
% *                                        command
% *         b) abort generation response - to respond to the abort generation
% *                                        command
% *
% *
% *      PROTOCOL DESCRIPTION:
% *
% *      1) The Controller Generation Server issues a "start generation command"
% *         to the RSS Signal Generation Manager.
% *
% *      2) The RSS Signal Generation Manager acknowledges the start generation
% *         command, and starts to generate the simulated radar signal data.
% *
% *         The RSS Signal Generation Manager calls svc_sendreply() to reply
% *         xdr_void to Controller Generation Server for acknowledgement of
% *         receiving start generation command.
% *
% *      3) During RSS signal generation, the RSS sends a "start generation
% *         response" message to the Controller under the following circumstances:
% *
% *         a. After generation of a period of time (to be determined by the RSS);
% *         b. when the RSS generation activity has detected a fatal error;
% *         c. at the completion of data generation activity.
% *
% *      4) The Controller acknowledges the response message.
% *         The Controller Server replies xdr_void using svc_sendreply().
% *
% *     5) The Controller Generation Server can issue an "abort generation
% *        command" to the RSS Signal Generation Manager to abort the
% *        generation activity for the current run, when the operator
% *        invokes ABORT command.
% *
% *     6) The RSS Signal Generation Manager acknowledges the abort generation
% *        command, and terminates the current generation.
% *        The Generation Manager replies xdr_void using svc_sendreply().
% *
% *     7) The RSS sends a "abort generation response" message to the Controller
% *        to indicate whether the generation is successfully aborted.
% *
% *     8) The Controller acknowledges the abort response.
% *        The Controller Server replies xdr_void using svc_sendreply().
% *
% *     9) The Controller can issue a "shutdown generation command"
% *        to the RSS Signal Generation Manager to shutdown RSS
% *        generation tasks and the RSS Generation Manager itself.
% *
% *    10) The RSS Signal Generation Manager acknowledges the shutdown
% *        command, and brings down all RSS signal generation processes and
% *        terminates itself.
% *        The Generation Manager replies xdr_void using svc_sendreply().
% *
% *    11) The Controller can issue a "provide status message" to the RSS
% *        Signal Generation Manager to ask for the current RSS generation
% *        status.
% *
% *    12) The RSS Signal Generation Manager replies the current RSS
% *        status t_crgi_RSS_state (running or idle) using svc_sendreply().
% *
% *    13) The Controller can issue a "ping message" to the RSS Signal
% *        Generation Manager to ask for an echo-response.
% *
% *    14) The RSS Signal Generation Manager replies xdr_void using
% *        svc_sendreply().
% */
#ifdef RPC_HDR
%#ifndef crgi_intf_h
%#define crgi_intf_h
%#include "spd_intf.h"

%/*-------------------------------------------------------------------------- *
% *                                                                           *
% *    DEFINE CONSTANTS                                                       *
% *                                                                           *
% *---------------------------------------------------------------------------*/
%/*
% * The RPC program and version definitions, which will be used to
% * register the services through the use of RPC svc_register call.
% *
% *      Program Number Range            Description
% *      ------------------------------------------------------------------
% *      0x20000000 to 0x200000ff        Controller server program number
% *      0x20000100 to 0x200001ff        RSS server program number
% *      0x20000200 t0 0x200002ff        ESP server program number
% *      0x20000300 t0 0x200003ff        DSP server program number
% *      0x20000400 t0 0x200004ff        RTSG server program number
% */
/* RPC program numver and version number registered by
   Controller Generation Server */
%#define c_crgi_RPCProg_CTL_GENSVR      ((u_long) 0x20000000)
%#define c_crgi_RPCVers_CTL_GENSVR      ((u_long) 1)

/* RPC program numver and version number registered by
   RSS Generation Manager */
%#define c_crgi_RPCProg_RSS_GENMGR      ((u_long) 0x20000100)
%#define c_crgi_RPCVers_RSS_GENMGR      ((u_long) 1)

%/*--------------------------------------------------------------------------
```

```
% *
% * The pathname (directory and file name) of the RSS temporary status/error
% * file logged by the RSS Generation Manager for a signal generation.
% * At the end of a signal generation run, the Controller Generation
% * Server will merge this file with the Controller status/error file
% * to form a combined generation error log file.
% *
% */
%#define c_crgi_log_file       "/home/trill/spdmgr/log/rss_gen_status.log"

#endif


%/*-----------------------------------------------------------------------------*
% *                                                                             *
% *    DEFINE ENUMERATED DATA TYPES AND MESSAGE STRUCTURES                      *
% *                                                                             *
% *-----------------------------------------------------------------------------*/

%/*================================================================
% *
% *   Controller ==> RSS direction message definitions.
% *
% *----------------------------------------------------------------
% *
% *   Integers enumerating command types sent to the RSS from the
% *   controller.
% */
enum t_crgi_gencmd_types
{
    c_crgi_start_gencmd,          /* start generation */
    c_crgi_abort_gencmd,          /* abort generation */
    c_crgi_shutdwn_gencmd,        /* shutdown generation */
    c_crgi_status_cmd,            /* status request */
    c_crgi_ping_cmd               /* ping message */
};

%/*----------------------------------------------------------------
% *
% *   Data structure for RSS Signal Generation Parameters
% *
% *   These parameters are passed with the start generation command to indicate
% *   which signal generation definitions are used for signal generation.
% *
% *      satdef_pathname  pathname (full directory and file name) of
% *                       Satellite/Radar Definition file.
% *                       If the file is stored in a remote disk, then
% *                       the disk must be remote-mounted via NFS for
% *                       RSS to read.
% *      envdef_pathname  pathname (full directory and file name) of
% *                       Environment Definition file.
% *                       If the file is stored in a remote disk, then
% *                       the disk must be remote-mounted via NFS for
% *                       RSS to read.
% *      SRSD_dir         output destination directory where RSS should write
% *                       Simulated Radar Signal Data (SRSD) file.
% *      prod_dir         output destination directory where RSS should write
% *                       intermediate products, RSS derived processing
% *                       parameters.
% *      gen_ident        signal generation identifier.
% *                       RSS adds subfix to gen_ident to generate the file name
% *                       for RSS intermediate result files, RSS generated
% *                       processing parameters and simulated radar signal
% *                       data (SRSD) file.
% *
% *      NOTE: RSS should store satdef_pathname and envdef_pathname in
% *            the header block of both RSS intermediate results, SRSD file
% *            and RSS derived processing parameters to indicate
% *            which definitions have been used to generate these output files.
% *
% */
struct t_crgi_gen_parms
{
   char satdef_pathname[c_spd_path_size];
                              /* Sat/Radar Definition file pathname */
   char envdef_pathname[c_spd_path_size];
                              /* Environment Definition file pathname */
   char SRSD_dir[c_spd_path_size];
                              /* SRSD file output directory */
   char prod_dir[c_spd_path_size];
                              /* Signal Generation output products
                                 directory */
   char gen_ident[c_spd_name_size];
                              /* Signal generation identifier */
};

%/*------------------------------------------------------------------
% *
% *   Data structure for Generation command message.
% *
% *   source:   Controller Simulation Server task
% *   sink:     RSS Generation Mananger task
% *
% *   The Generation Command message consists of a command type and command
% *   parameters.  The command type tells which command parameters are
% *   sent from the Controller to the RSS.
% *
% *   The Generation command type can be either:
% *
% *       1) start_gencmd       - start generation parameters follow
% *                               in the command message
% *       2) abort_gencmd       - no parameters follow
% *       3) shutdwn_gencmd     - no parameters follow
% *       4) status_cmd         - no parameters follow
% *       5) ping_cmd           - no parameters follow
% */
union t_crgi_gencmd switch (t_crgi_gencmd_types command_type)
{
   case c_crgi_start_gencmd:          /* start generation message */
      t_crgi_gen_parms            gen_parms;

   case c_crgi_abort_gencmd:          /* abort generation message */
      void;

   case c_crgi_shutdwn_gencmd:        /* shutdown generation message */
      void;

   case c_crgi_status_cmd:            /* status request message */
      void;

   case c_crgi_ping_cmd:              /* ping message */
      void;
};


%/*==================================================================
% *
% *   RSS ==> Controller direction message definitions.
% *
% *------------------------------------------------------------------
% *
% *   Integers enumerating RSS generation state types
% */
enum t_crgi_RSS_state
{
   c_crgi_running,            /* generation is currently running */
   c_crgi_idle                /* generation is idle */
};

%/*------------------------------------------------------------------
```

```
% *
% *   Integers enumerating response types sent from the RSS in response
% *   to processing a generation command.
% */
enum t_crgi_genrsp_type
{
   c_crgi_start_genrsp,         /* RSS response for start generation */
   c_crgi_abort_genrsp          /* RSS response for abort generation */
};

%/*-----------------------------------------------------------------------
% *
% *   Integers enumerating status return codes from the RSS for
% *   the current signal generation activity.
% */
enum t_crgi_genstat_type
{
   c_crgi_in_progress,          /* generation activity still in progress  */
   c_crgi_gen_compl,            /* generation activity completed normally */
   c_crgi_gen_failed            /* generation activity terminated with error */
};

%/*-----------------------------------------------------------------------
% *
% *   Data structure for RSS start generation status
% *
% *   source:   RSS Generation Manager Task
% *   sink:     Controller Generation Server Task
% *
% *   RSS start generation status consists of:
% *
% *      status               -  indicates whether the generation activity is
% *                              currently in progress, completed normally, or
% *                              terminated with error.
% *      percent              - percent completion of the current
% *                              RSS radar signal generation
% *
% *      In all cases (normally completed, failed, or in progress),
% *      the percent completion must be filled by RSS and send back
% *      to the Controller.
% *      The percent complete status is sent from the RSS to the Controller
% *      on a periodic basis (to be determined by the RSS) to indicate
% *      the progress of generation activity.
% *      At normal completion, it implies 100%; when terminated with
% *      error, it implies the percent complete at termination.
% *
% *
% */
struct t_crgi_gen_status
{
   t_crgi_genstat_type  status;        /* current RSS signal generation status */
   t_spd_real           percent;       /* current percent complete */
};

%/*-----------------------------------------------------------------------
% *
% *   Enumerated data type for RSS abort Generation status
% *
% *   source:   RSS Generation Manager Task
% *   sink:     Controller Generation Server Task
% *
% *   RSS abort generation status indicates whether the current generation
% *   activity can be either successfully aborted or aborted with error.
% */
enum t_crgi_abort_status
{
   c_crgi_abort_compl,                    /* successfully aborted the
                                             current run  */
   c_crgi_abort_failed                    /* failed to abort generation */

};

%/*-----------------------------------------------------------------
% *
% *   Data structure for Generation responses message.
% *
% *   source:   RSS Generation Manager Task
% *   sink:     Controller Generation Server Task
% *
% *   The Generation Response message consists of a response type and response
% *   parameters.  The response type tells which response parameters are
% *   sent from the RSS to the Controller.
% *
% *   The Generation Response command type can be either:
% *
% *       1) start_genrsp - generation status parameters follow
% *       2) abort_genrsp - abort status parameters follow
% *
% */
union t_crgi_genrsp switch (t_crgi_genrsp_type rsp_type)
{
   case c_crgi_start_genrsp:
      t_crgi_gen_status         gen_status;     /* start generation response */

   case c_crgi_abort_genrsp:
      t_crgi_abort_status       abort_status;   /* abort generation response */
};

#ifdef RPC_HDR
%#endif
#endif
```

THIS PAGE INTENTIONALLY LEFT BLANK

D1.5 RSS Playback Interface (CRPI)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*=======================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      crpi_intf.x - Controller/RSS Simulation Interface

                CRPI is a package under the RIF [RSS InterFace]
                CSC in the Controller functional area. CRPI Specifies the
                Controller/RSS interface for Simulation Run
                Activities only.


Identification: [@(#)]

SCCS Id:        @(#)crpi_intf.x 1.4 93/07/09

Responsible Organization:

                MacDonald Dettwiler & Associates

Purpose:        This package declares RPC interfaces (constants and message
                types) between the Simulation Server task in Controller
                functional area and the RSS Playback Manager task in the
                RSS functional area for a simulation run activity.  The
                message definitions and protocol are based on the Testbed
                Preliminary Design Document.

                This file is written in RPC data description language in XDR
                (eXternal Data Representation) standard. This file will use
                "rpcgen" utility to generate the following two files:

                crpi_intf.h     - global include file for both Controller
                                  and RSS simulation task packages to include;
                crpi_intf_xdr.c - XDR source file which contains required XDR
                                  conversion routines to support the RPC
                                  message data types.

                The generated files are needed for building the executables for
                both Controller Simulation Server and RSS Playback Manager Task.

Designer:       Rob Gottschalk

Change History:

    SCR   Date          Author    Change made
    ---   -----------   ------    ------------------------------------------
    n/a   06-Jan-1993   robg     Initial Release
          04-Feb-1993   cst      overall update; add protocol description
          10-Mar-1993   cst      after 40% review
                                 changed c_crpi_completed_normally to
                                         c_crpi_sim_compl
                                         c_crpi_terminated_with_error to
                                         c_crpi_sim_failed
                                         c_crpi_abort_completed to
                                         c_crpi_abort_compl
                                         *playback_command to *plbcmd
                                         *playback_rsp to *plbrsp
                                 protocol changed by adding c_crpi_status_cmd
                                    and c_crpi_ping_cmd
                                 added t_crpi_RSS_state
          03-May-1993   cst      added c_crpi_spmt_lognam (filename of RSS
                                 status/error file)
          22-Jun-1993   cst      replaced c_crpi_spmt_lognam by
                                   c_crpi_log_file to include directory name
                                   (Design Issue #28)

=====================================================================*/
```

```

%/*
% *
% *     PROCESS STARTUP/SHUTDOWN
% *
% *     RSS Signal Playback Manager works as a server (daemon process)
% *     for Controller Simulation Task, while the Controller Simulation
% *     Task is a server (daemon process) serving Simulation Run HMI tasks.
% *     the Controller Simulation Server and the RSS Signal Playback Manager
% *     will be running all the time and await for playback requests either
% *     from the operator and from a batch program.
% *
% *     Both Controller Simulation Server and RSS Playback Manager will
% *     be launched or shutdown by the SPD System Manager through a script file
% *     in which an offline program will be invoked. (See package CSYS).
% *
% *     The Controller would send a RSS shutdown message to RSS Playback
% *     Manager to ask for shutdown. RSS should gracefully bring down all
% *     related child and daemon processes for signal playback activity
% *     in RSS and then terminate the RSS Playback Manager itself.
% *
% *     MESSAGE TYPES:
% *
% *     1) Command messages: sent from the Controller to RSS
% *
% *        Asynchronous message:
% *
% *        a) start playback command - to start a playback activity
% *        b) abort playback command - to stop the current playback activity
% *
% *        Synchronous message:
% *
% *        c) shutdown playback command - to shutdown RSS playback processes
% *        d) provide status message - request the current playback status
% *                                    (running or idle)
% *        e) ping message            - request an echo-response
% *
% *     2) Response messages: sent from RSS to the Controller
% *
% *        a) start playback response - to respond to the start playback command
% *        b) abort playback response - to respond to the abort playback command
% *
% *        NOTE: Assume that RSS will shutdown all playback processes including
% *              the RSS Playback Manager, there is no response required from
% *              the RSS to the Controller in response to the shutdown playback
% *              command.
% *
% *
% *     PROTOCOL DESCRIPTION:
% *
% *     1) The Controller Simulation Server issues a "start playback command"
% *        to the RSS Playback Manager.
% *
% *     2) The RSS Playback Manager acknowledges the start command,
% *        and starts to playback the simulated radar signal data.
% *
% *     3) During playback, the RSS sends a "start playback response" message
% *        to the Controller under the following circumstances:
% *
% *        a. After playback of each iteration of the data segment;
% *        b. when the RSS playback activity has detected a fatal error;
% *        c. at the completion of data playback activity.
% *
% *     4) The Controller acknowledges the response message.
% *
% *     5) The Controller Simulation Server can issue an "abort playback command"
% *        to the RSS Playback Manager to abort the playback activity for
% *        the current run, when the operator invokes ABORT command or the
% *        Controller detects DSP failure.
% *
% *    6) The RSS Playback Manager acknowledges the abort playback command, and
% *       terminates the current playback.
% *
% *    7) The RSS sends a "abort playback response" message to Controller to
% *       indicate whether the playback is successfully aborted.
% *
% *    8) The Controller acknowledges the abort response.
% *
% *    9) The Controller Simulation Server can issue a "shutdown playback
% *       command" to the RSS Signal Playback Manager to shutdown RSS playback
% *       tasks.
% *
% *   10) The RSS Playback Manager replies xdr_void using svc_sendreply(),
% *       and brings down all RSS signal playback child processes and
% *       terminates itself.
% *
% *   11) The Controller can issue a "provide status message" to the RSS
% *       Signal Playback Manager to ask for the current RSS playback
% *       status.
% *
% *   12) The RSS Signal Playback Manager replies the current RSS
% *       state t_crpi_RSS_state (running or idle) using svc_sendreply()
% *
% *   13) The Controller can issue a "ping message" to the RSS Signal
% *       Playback Manager to ask for an echo-response.
% *
% *   14) The RSS Signal Playback Manager replies xdr_void using
% *       svc_sendreply().
% */

#ifdef RPC_HDR
%#ifndef crpi_intf_h
%#define crpi_intf_h
%#include "spd_intf.h"

%/*------------------------------------------------------------------------- *
% *                                                                          *
% *    DEFINE CONSTANTS                                                      *
% *                                                                          *
% *-------------------------------------------------------------------------*/
%/*
% * The RPC program and version definitions, which will be used to
% * register the services through the use of RPC svc_register call.
% *
% *      Program Number Range            Description
% *      ---------------------------------------------------------------
% *      0x20000000 to 0x200000ff        Controller server program number
% *      0x20000100 to 0x200001ff        RSS server program number
% *      0x20000200 t0 0x200002ff        ESP server program number
% *      0x20000300 t0 0x200003ff        DSP server program number
% *      0x20000400 t0 0x200004ff        RTSG server program number
% *
% */
/* RPC program numver and version number registered by
   Controller Simulation Server */
%#define c_crpi_RPCProg_CTL_SIMSVR      ((u_long) 0x20000001)
%#define c_crpi_RPCVers_CTL_SIMSVR      ((u_long) 1)

/* RPC program numver and version number registered by
   RSS Playback Manager */
%#define c_crpi_RPCProg_RSS_PLBMGR      ((u_long) 0x20000101)
%#define c_crpi_RPCVers_RSS_PLBMGR      ((u_long) 1)

%/*-----------------------------------------------------------------------
% *
% * The pathname (directory and file name) of the temporary RSS status/error
% * file logged by the RSS Playback Manager for a simulation run.
% * At the end of a simulation run, the Controller Simulation Server
% * will merge this file with ESP status/error file, RSP status/error file,
% * and the Controller simulation status/error file (temporary error files)
% * to form a conbined simulation error log file.
% *
% */
%#define c_crpi_log_file        "/home/trill/spdmgr/log/rss_sim_status.log"
#endif

%/*------------------------------------------------------------------------ *
% *                                                                          *
% *    DEFINE ENUMERATED DATA TYPES AND MESSAGE STRUCTURES                   *
% *                                                                          *
% *------------------------------------------------------------------------*/

%/*================================================================
% *
% *   Controller ==> RRS direction message definitions.
% *
% *----------------------------------------------------------------
% *
% *   Integers enumerating command types sent to the RSS from the
% *   controller.
% */
enum t_crpi_simcmd_types
{
   c_crpi_start_plbcmd,                 /* start playback */
   c_crpi_abort_plbcmd,                 /* abort playback */
   c_crpi_shutdwn_plbcmd,               /* shutdown playback */
   c_crpi_status_cmd,                   /* status request */
   c_crpi_ping_cmd                      /* ping message */
};

%/*----------------------------------------------------------------
% *
% *   Data structure for RSS Playback Parameters
% *
% *   These parameters are passed with the start playback command to indicate
% *   which simulated radar signal data (SRSD) is playing back, and how many
% *   iterations are used.
% *
% *      SRSD_pathname  - pathname (full directory name and file name)
% *                       which indicates where the SRSD file is stored.
% *                       If the file is stored in a remote disk, then
% *                       the disk must be remote-mounted via NFS.
% *                       RSS uses the provided SRSD_pathname file name
% *                       to playback the simulated signal data.
% *
% *      NOTE: since RSS does not need simulation run definition for playback,
% *            no simulation run definition identifier or pathname will be
% *            passed to RSS.
% *
% *      iteration_count      -  number of iterations used for a run
% *
% */
struct t_crpi_playback_parms
{
   char         SRSD_pathname[c_spd_path_size]; /* pathname of SRSD file */
   t_spd_int32  iteration_count;                /* number of interations */
};

%/*----------------------------------------------------------------
% *
% *   Data structure for Playback command message.
% *
% *   source:   Controller Simulation Server task
% *   sink:     RSS Playback Mananger task
% *
% *   The Playback Command message consists of a command type and command
% *   parameters.  The command type tells which command parameters are
% *   sent from the Controller to the RSS.
% *
% *   The Playback command type can be either:
```

```
% *
% *     1) start_plbcmd      - RSS Playback parameters follow
% *                            in the command message
% *     2) abort_plbcmd      - no parameters follow
% *     3) shutdwn_plbcmd    - no parameters follow
% *     4) status_cmd        - no parameters follow
% *     5) ping_cmd          - no parameters follow
% */
union t_crpi_plbcmd switch (t_crpi_simcmd_types command_type)
{
   case c_crpi_start_plbcmd:            /* start playback parameters */
      t_crpi_playback_parms playback_parms;

   case c_crpi_abort_plbcmd:            /* abort playback command */
      void;

   case c_crpi_shutdwn_plbcmd:          /* shutdown playback command */
      void;

   case c_crpi_status_cmd:              /* status request message */
      void;

   case c_crpi_ping_cmd:                /* ping message */
      void;
};


%/*=====================================================================
% *
% *  RSS ==> Controller direction message definitions.
% *
% *---------------------------------------------------------------------
% *
% *  Integers enumerating RSS state types
% */
enum t_crpi_RSS_state
{
   c_crpi_running,              /* playback is currently running */
   c_crpi_idle                  /* playback is idle */
};

%/*--------------------------------------------------------------------
% *
% *  Integers enumerating response types sent from the RSS in response
% *  to processing a playback command.
% */
enum t_crpi_plbrsp_types
{
   c_crpi_start_plbrsp,         /* RSS response for start playback */
   c_crpi_abort_plbrsp          /* RSS response for abort playback */
};

%/*--------------------------------------------------------------------
% *
% *  Integers enumerating status return codes from the RSS for
% *  the current playback activity.
% */
enum t_crpi_simstat_types
{
   c_crpi_in_progress,          /* playback activity still in progress  */
   c_crpi_sim_compl,            /* playback activity completed normally */
   c_crpi_sim_failed            /* playback activity terminated with error */
};

%/*--------------------------------------------------------------------
% *
% *  Data structure for RSS playback status
% *
% *  source:   RSS Playback Manager Task
% *  sink:     Controller Simulation Server Task
% *
% *   RSS playback status consists of:
% *
% *      status               - indicates whether the playback activity is
% *                              currently in progress, completed normally, or
% *                              terminated with error.
% *      iteration_no         - iteration number of data segment just completed
% *                              data playback
% *      elapsed_time         - actual elapsed time took by RSS for playing
% *                              back radar signal upto the last iteration
% *                              or the elapsed time took upto termination
% *                              (successful or fatal)
% *
% *      In all cases (normally completed, failed, or in progress),
% *      the iteration number and elapsed time must be filled by RSS and
% *      send back to the Controller. If termination occurred before
% *      completing the first iteration, the iteration number is set to 0.
% */
struct t_crpi_playback_status
{
   t_crpi_simstat_types status; /* current RSS simulation playback status */
   t_spd_int32  iteration_no;   /* last iteration in the playback activity */
   t_spd_real   elapsed_time;   /* indicates the elapsed time of data
                                   playback for the last iteration
                                   (interation_no) */

};

%/*-------------------------------------------------------------------------
% *
% *   Enumerated data type for RSS abort playback status
% *
% *   source:   RSS Playback Manager Task
% *   sink:     Controller Simulation Server Task
% *
% *   RSS abort playback status indicates whether the current playback
% *   activity can be either successfully aborted or aborted with error.
% */
enum t_crpi_abort_status
{
   c_crpi_abort_compl,          /* successfully aborted the current run  */
   c_crpi_abort_failed          /* failed to abort playback */

};

%/*-------------------------------------------------------------------------
% *
% *
% *   Data structure for Playback Responses message.
% *
% *   source:   RSS Playback Manager Task
% *   sink:     Controller Simulation Server Task
% *
% *   The Playback Response message consists of a response type and response
% *   parameters.  The response type tells which response parameters are
% *   sent from the RSS to the Controller.
% *
% *   The PLayback Response response type can be either:
% *
% *       1) start_plbrsp - playback status parameters follow
% *       2) abort_plbrsp - abort status parameters follow
% *
% */
union t_crpi_plbrsp switch (t_crpi_plbrsp_types rsp_type)
{
   case c_crpi_start_plbrsp:
      t_crpi_playback_status    playback_status;/* start playback response */

   case c_crpi_abort_plbrsp:
      t_crpi_abort_status       abort_status;   /* abort playback response */
};
```

```

#ifdef RPC_HDR
%#endif
#endif
```

THIS PAGE INTENTIONALLY LEFT BLANK

D1.6 DSP Simulation Interface (CDSI)

The source file of this CSU (cdsi_intf.x) is provided in Appendix A of the DSP/Testbed Interface Detailed Design document [R-1].

THIS PAGE INTENTIONALLY LEFT BLANK

D1.7 Data Analysis (CDAT)

Source files for the intermediate products ingest procedures in PV-Wave includes:

a) DSP Processing Parameters:

cdat_spd_intf.pro
cdat_dspparm_intf.pro
cdat_dspparm_rd.pro

b) DSP Intermediate Results:

cdat_dspir_intf.Pro
cdat_dspir_rd.pro

c) RSS Intermediate Results:

cdat_rrsir_intf.pro
cdat_rssir_rd.pro

THIS PAGE INTENTIONALLY LEFT BLANK

```
; ============================================================================
;
; SPD - Signal Processing Demonstrator Computer System
;
; (C) Copyright Her Majesty the Queen in Right of Canada
; as represented by the Minister of National Defence
;
; File Name:    cdat_spd_intf.pro - SPD global include file
;
; Identification:       [@(#)]
;
; SCCS Id:      @(#)cdat_spd_intf.pro   1.1 93/09/17
;
; Responsible Organization:
;
;               MacDonald Dettwiler and Associates
;
; Purpose:      This file declares constants and data structures for
;               the SPD system in PV_Wave language.
;
;               This package is depending on spd_intf.x
;
; Designer:     Stephen Tran
;
; Change History:
;
;    SCR  Date         Author   Change made
;    ---  -----------  ------   ----------------------------------------
;    n/a 01-Sep-1993  cst     Initial Release
;
; ============================================================================
; INCLUDE FILE cdat_spd_intf.pro
;
;   Description:
;       This file contains constants and data structures defined in
;       PV-Wave language.
;       These constants and data structures were originally declared
;       in SPD package (see spd_intf.x).
;       These include generic constants, generic data type and data structures.
;
;   Synopsis:
;       @cdat_spd_intf
;
;       NOTE: this file must be included in the main program, before
;             calling any procedure or function provided by this package.
;
; ============================================================================
;              I M P L E M E N T A T I O N
; ============================================================================
;
; Define constants
;
c_spd_max_DI=64         ; max number of dwell intervals
c_spd_max_NCI=12        ; max number of non-coherent integration
c_spd_max_CPI=64        ; max number of coherent processing intervals
c_spd_max_ph_ctrs=3     ; max number of phase centres per DPCA group
c_spd_max_DPCA_grps=2   ; max number of DPCA groups
c_spd_max_tot_ph_ctrs=c_spd_max_ph_ctrs*c_spd_max_DPCA_grps
                        ; max total number of phase centres used for DPCAs
c_spd_max_aux=16        ; max number of auxiliaries for nulling blanking or both
c_spd_max_ECCM_DOF=16   ; max number of ECCM degree of freedom or
                        ; total number of auxiliaries and taps
c_spd_max_nulling_chan=c_spd_max_ECCM_DOF-1
                        ; max number of nulling channels for blanker
c_spd_max_taps=3        ; max number of delay taps for each auxiliary
c_spd_max_blkrs=4       ; max number of blankers
c_spd_max_rng_dop_blkrs=2
                        ; max number of range-Doppler blankers
c_spd_max_chan=c_spd_max_tot_ph_ctrs + c_spd_max_ECCM_DOF
                        ; max number of ESP input channels
```

```
c_spd_max_weights=7     ; max number of weights for user-defined
                        ; notch filter weights
c_spd_max_elm_arbtr_arr=16      ; max number of elements for arbitrary array
c_spd_max_formed_beam=2 ; max number of formed beam
c_spd_max_RSP_chan=3    ; max number of RSP input channels
c_spd_max_nulled_chan=c_spd_max_tot_ph_ctrs + c_spd_max_blkrs
                        ; max number of nulled channels
c_spd_max_blanked_chan=6 ; max number of PD-blanked channels
c_spd_max_CC_pairs=2    ; max number of ASAR DPCA clutter cancellation pairs
c_spd_max_nulling_sys= c_spd_max_formed_beam + c_spd_max_blkrs
                        ; max number of nulling systems
c_spd_tbl_name=30       ; max user-entered table name
c_spd_name_size=12      ; max definition id length
c_spd_path_size=256     ; max path name (directory + filename) length
c_spd_file_size=128     ; max file name length
c_spd_max_msglen=256    ; max message text length
c_spd_num_SPB_RTSG_DSP=2        ; max number of synchronous parallel
                                ; bus from RTSG to DSP
c_spd_num_SPB_ESP_RSP=2 ; max number of synchronous parallel
                        ; bus from ESP to RSP
c_spd_sw_vers_id_len=32 ; length of unit (ESP or RSP) software version
                        ; identifier
c_spd_replica_size=512  ; replica array size
;
; Define structures
;
TMP={t_spd_complexD, $
          real_d: 0.0D, imag_d: 0.0D}
TMP={t_spd_decimation, $
          start:0L, stop:0L, step:0L}

; release data memory
TMP=0L

```

```
; ==========================================================================
;
; SPD - Signal Processing Demonstrator Computer System
;
; (C) Copyright Her Majesty the Queen in Right of Canada
; as represented by the Minister of National Defence
;
; File Name:     cdat_dspparm_intf.pro - DSP Processing Parameters include file
;
; Identification:        [@(#)]
;
; SCCS Id:       @(#)cdat_dspparm_intf.pro       1.1 93/09/17
;
; Responsible Organization:
;
;                MacDonald Dettwiler and Associates
;
; Purpose:       This file declares constants and data structures for
;                ingesting DSP Processing Parameters file in PV_Wave language.
;
;                The DSP Processing Parameters data file is stored in XDR format.
;                This package is depending on cdsi_intf.x
;
;                Since cdsi_intf.x is depending on spd_intf.x, before loading
;                this include file into PV-Wave cdat_spd_intf.pro (a PV-Wave
;                version of spd_intf.x) must be loaded first.
;
; Designer:      Stephen Tran
;
; Change History:
;
;    SCR  Date        Author    Change made
;    ---  ----------- ------   ----------------------------------------
;    n/a 01-Sep-1993  cst      Initial Release
;
; ==========================================================================
;
; INCLUDE FILE cdat_dspparm_intf.pro
;
;   Description:
;       This file contains constants and data structures defined in
;       PV-Wave language for DSP parameters.
;       The DSP parameters data structure is defined in package CDSI
;       (cdsi_intf.x). This is a PV-Wave version of cdsi_intf.x.
;
;   Synopsis:
;       @cdat_dspparm_intf
;
;       NOTE: this file must be included first in the main program, before
;             calling the DSP processing parameters ingest functions
;             provided by this package.
;
; ==========================================================================
;                I M P L E M E N T A T I O N
; ==========================================================================
;
; Define constants
;
c_cdsi_sw_vers_id_len = c_spd_sw_vers_id_len
c_cdsi_replica_size = c_spd_replica_size
;
; Define structures
;
t_cdsi_sw_vers_id=BYTARR(c_cdsi_sw_vers_id_len)
t_cdsi_PE_sw_image=BYTARR(c_spd_file_size)
pathname=BYTARR(c_spd_path_size)
id_name=BYTARR(c_spd_name_size)

TMP={t_cdsi_CPI_waveform, $
          RF_freq:0.0, $
```

```
                rx_tm_start_burst:0.0, $
                CPI_samplg_ofst:0.0}

TMP={t_cdsi_CPI_elm, $
                platfm_rad_vel:0.0, $
                earth_rad_vel:0.0, $
                num_samp_input:0L, $
                waveform:{t_cdsi_CPI_waveform}, $
                capture_CPI_enable:0L}

TMP={t_cdsi_NCI_elm, $
                CPI_len:0L, $
                CPI_val:REPLICATE({t_cdsi_CPI_elm}, c_spd_max_CPI), $
                inter_ph_ctr_sep:0.0, $
                samples_per_PRI:0L, $
                num_pulses_tx:0L, $
                num_pulses_proc:0L, $
                num_bursts_integ:0L, $
                OS_NCI_rank:0L, $
                capture_NCI_enable:0L}

TMP={t_cdsi_engag_geometry, $
                platfm_heading:0.0, $
                look_depression:0.0, $
                look_azimuth:0.0, $
                ref_range:0.0}

TMP={t_cdsi_pulse_doppler, $
                PD_FFT_len:0L, $
                doppler_aug_sel:0L, $
                unambig_PRF:0.0}

TMP={t_cdsi_ASAR_DPCA, $
                num_DPCA_cells:0L, $
                num_cc_rng_gates:0L, $
                rng_gate_type:0L, $
                start_rng_gate:0L, $
                step_rng_gate:0L}

TMP={t_cdsi_detect_thrshd, $
                doppler_width:0L, $
                range_width:0L, $
                guard_doppler_width:0L, $
                guard_range_width:0L, $
                OS_CFAR_rank:0.0, $
                prob_false_alarm:0.0}


TMP={t_cdsi_parm_est, $
                num_PRF_used:0L, $
                footprint_len:0.0, $
                RCS_scale_factor:0.0}

TMP={t_cdsi_DI_waveform, $
                pulse_replica:COMPLEXARR(c_cdsi_replica_size), $
                replica_len:0L, $
                exp_pulse_dur:0.0, $
                compr_pulse_width:0.0}

TMP={t_cdsi_DI_elm, $
                NCI_len:0L, $
                NCI_val:REPLICATE({t_cdsi_NCI_elm}, c_spd_max_NCI), $
                engag_geom:{t_cdsi_engag_geometry}, $
                noise_power_main:0.0, $
                noise_power_aux:0.0, $
                waveform:{t_cdsi_DI_waveform}, $
                PD:{t_cdsi_pulse_doppler}, $
                ASAR_DPCA:{t_cdsi_ASAR_DPCA}, $
                detect_thrshd:{t_cdsi_detect_thrshd}, $
                parm_est:{t_cdsi_parm_est}, $
```

```
             capture_DI_enable:0L}

TMP={t_cdsi_xy, $
             x:0.0, $
             y:0.0}

TMP={t_cdsi_antenna, $
            num_DPCA_grps:0L, $
            num_ph_ctr_per_grp:0L, $
            PRI_multiple:0L, $
            disp:REPLICATE({t_cdsi_xy}, c_spd_max_DPCA_grps), $
            num_aux:0L, $
            num_taps:LONARR(c_spd_max_taps)}

TMP={t_cdsi_general_config, $
            DPCA_mode:0L, $
            monopulse_enable:0L, $
            num_rg_dop_blankers:0L, $
            MASR_enable:0L}

TMP={t_cdsi_ph_ctr_nulling_chan, $
            num_nulling_chan:0L, $
            nulling_chan:LONARR(c_spd_max_ECCM_DOF)}

TMP={t_cdsi_blanker_nulling_chan, $
            num_nulling_chan:0L, $
            nulling_chan:LONARR(c_spd_max_nulling_chan)}

TMP={t_cdsi_ECCM_config, $
            num_MASR_ph_ctr:0L, $
            monopulse_lobe_sel:0L, $
            DPCA_group_sel:0L, $
            ph_ctr_pair_sel:LONARR(c_spd_max_DPCA_grps), $
            ph_ctr_nulling: $
                REPLICATE({t_cdsi_ph_ctr_nulling_chan},c_spd_max_tot_ph_ctrs)}

TMP={t_cdsi_ECCM_proc, $
            num_pre_dop_blankers:0L, $
            blanker_chan:LONARR(c_spd_max_blkrs), $
            blanker_nulling: $
                  REPLICATE({t_cdsi_blanker_nulling_chan}, c_spd_max_blkrs), $
            predop_blk_noise_thrshd:0.0, $
            predop_blk_slope_thrshd:0.0, $
            predop_blk_channel_comb:0L, $
            snapshot_smooth_period:0.0, $
            num_snapshots:0L, $
            num_update_intervals:0L, $
            snapshot_selection:0L, $
            snapshot_edit:0L, $
            num_snapshots_rejected:0L, $
            notch_filter_len:0L, $
            notch_filter_type:0L, $
            notch_filter_weights:COMPLEXARR(c_spd_max_weights), $
            diag_loading_const:0.0, $
            stability_thrshd:0.0, $
            weight_size_thrshd_enable:0L, $
            weight_size_thrshd:0.0, $
            MASR_mode:0L}

TMP={t_cdsi_radar_proc, $
            SC_enable:0L, $
            SC_filter_len:0L, $
            SC_filter_update_int:0L, $
            freq_align_enable:0L, $
            PD_enable:0L, $
            PD_window_type:0L, $
            PD_kaiser_SL_lvl:0.0, $
            pulse_compr_enable:0L, $
            doppler_comp_enable:0L, $
            PC_window_type:0L, $
```

```
           PC_kaiser_SL_lvl:0.0, $
           ASAR_DPCA_enable:0L, $
           deweight_TIR:0.0, $
           rgdop_blk_noise_thrshd:0.0, $
           rgdop_blk_slope_thrshd:0.0, $
           rgdop_blk_channel_comb:0L, $
           integ_method:0L, $
           target_rng_walk_comp_enable:0L, $
           sampl_align_enable:0L, $
           vel_align_enable:0L, $
           nominal_carrier_freq:0.0, $
           nominal_PRF:0.0, $
           detect_type:0L}

TMP={t_cdsi_IR_capture, $
           capt_ESP_input:LONARR(c_spd_max_chan), $
           capt_post_SC:LONARR(c_spd_max_chan), $
           capt_nulled:LONARR(c_spd_max_nulled_chan), $
           capt_ESP_blanked:LONARR(c_spd_max_blanked_chan), $
           capt_post_MASR:LONARR(c_spd_max_formed_beam), $
           capt_post_FA:LONARR(c_spd_max_RSP_chan), $
           capt_post_PD:LONARR(c_spd_max_RSP_chan), $
           capt_RSP_input:LONARR(c_spd_max_RSP_chan), $
           capt_post_PC:LONARR(c_spd_max_RSP_chan), $
           capt_post_CC:LONARR(c_spd_max_CC_pairs), $
           capt_monopulse_angle:0L, $
           capt_RCS:0L, $
           capt_RSP_blanked:0L, $
           capt_post_NCI:0L, $
           capt_weight_vectors:LONARR(c_spd_max_nulling_sys), $
           capt_R_r:LONARR(c_spd_max_nulling_sys), $
           capt_U_z:LONARR(c_spd_max_nulling_sys), $
           capt_snapshots:LONARR(c_spd_max_nulling_sys), $
           capt_PD_window:0L, $
           capt_DPCA_weights:0L, $
           capt_target_deweight:0L, $
           capt_PC_replicas:0L}

TMP={t_cdsi_signal_reduction, $
           time_decim:{t_spd_decimation}, $
           range_decim:{t_spd_decimation}, $
           doppler_decim:{t_spd_decimation}, $
           aug_doppler_decim:{t_spd_decimation}, $
           magnitude_sqrd:0L, $
           capture_CTI_enable:0L}

TMP={t_cdsi_DSP_process_parms, $
           SRSD_pathname:pathname, $
           RSSproc_pathname:pathname, $
           simdef_pathname:pathname, $
           IR_dir:pathname, $
           run_ident:id_name, $
           altitude:0.0, $
           antenna:{t_cdsi_antenna}, $
           sampling_rate:0.0, $
           pulse_type:0L, $
           DI_len:0L, $
           DI_val:REPLICATE({t_cdsi_DI_elm}, c_spd_max_DI), $
           general_config:{t_cdsi_general_config}, $
           ECCM_config:{t_cdsi_ECCM_config}, $
           ECCM_proc:{t_cdsi_ECCM_proc}, $
           radar_proc:{t_cdsi_radar_proc}, $
           IR:{t_cdsi_IR_capture}, $
           sig_reduct:{t_cdsi_signal_reduction} }

TMP={t_dspparm_prod_handle, $
           file_name:"", $
           unit:0L }

; release data memory
```

```
TMP=0L
```

```
; ============================================================================
;
; SPD - Signal Processing Demonstrator Computer System
;
; (C) Copyright Her Majesty the Queen in Right of Canada
; as represented by the Minister of National Defence
;
; File Name:    cdat_dspparm_rd.pro - Read DSP parameters file into memory
;                                     (coded in PV-Wave command language)
;
; Identification:       [@(#)]
;
; SCCS Id:      @(#)cdat_dspparm_rd.pro 1.3 93/09/17
;
; Responsible Organization:
;
;               MacDonald Dettwiler and Associates
;
; Purpose:      This file implements ingest functions in PV-Wave command
;               language to read DSP parameters file into memory.
;
;               The DSP parameters file is written out using the file format
;               described in the cdsi package.
;
; Designer:     Stephen Tran
;
; Change History:
;
;    SCR  Date         Author    Change made
;    ---  -----------  ------    ------------------------------------------
;    ??? 07-May-1993  cst      Initial Release
;
; ============================================================================
; USAGE:  This describes how to use the procedures or functions
;         provided by this module.
;
; when on "wave" prompt in the PV-Wave command line, type:
;      wave> .run, cdat_dspparm_rd.pro            - compile cdat_dspparm_rd.pro
;      wave> @cdat_spd_intf                       - include spd includes
;      wave> @cdat_dspparm_intf                   - include cdsi includes
;      wave>
;      wave> dspparm_open, "file_name", handle    - open an XDR file
;      wave> dsp_parm = dspparm_rd (handle)       - read DSP processing parameters
;      wave> dspparm_close, handle                - close XDR file
;      wave> dspparm_print, dsp_parm              - print out DSP processing
;                                                   parameters
;
; ============================================================================
;
;     R E A D   U T I L I T Y   F U N C T I O N S
;
; ============================================================================
; FUNCTION:    DSPPARM_RD_STRING
;
;   Description:
;       This function reads a string from an XDR file. The string size in
;       characters is passed to this function.  The string size can
;       be known by applying the "SIZE" function to the string.
;       The returned string is defined as a byte array with the specified
;       string size.
;
;   Synopsis:
;       string = DSPPARM_RD_STRING (prod_handle, str_len)
;
;   Parameters:
;       prod_handle            - in    - product handle returned from
;                                        DSPPARM_OPEN
;       str_len                - in    - string size in characters
;       string                 -   out - a string defined in byte array
;
FUNCTION DSPPARM_RD_STRING, prod_handle, str_len
```

```
   str=BYTARR(str_len)
   char=0L
   FOR i=0, str_len-1 DO BEGIN
      READU, prod_handle.unit, char
      str(i)=BYTE(char,3)
   END
   RETURN, str
END

; ========================================================================
; FUNCTION:    DSPPARM_RD_FLOAT
;
;   Description:
;       This function reads a single-precision floating point value
;       from an XDR file.
;
;   Synopsis:
;       float_val = DSPPARM_RD_FLOAT (prod_handle)
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         DSPPARM_OPEN
;       float_val               -   out - floating pointing value
;
FUNCTION DSPPARM_RD_FLOAT, prod_handle
   float_val=0.0
   READU, prod_handle.unit, float_val
   RETURN, float_val
END

; ========================================================================
; FUNCTION:    DSPPARM_RD_LONG
;
;   Description:
;       This function reads a long integer (32 bytes)
;       from an XDR file.
;
;   Synopsis:
;       lonint_val = DSPPARM_RD_LONG (prod_handle)
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         DSPPARM_OPEN
;       lonint_val              -   out - long integer value
;
FUNCTION DSPPARM_RD_LONG, prod_handle
   long_val=0L
   READU, prod_handle.unit, long_val
   RETURN, long_val
END


; ========================================================================
;
;     O P E N / C L O S E    F U N C T I O N S
;
; ========================================================================
; PROCEDURE:    DSPPARM_OPEN
;
;   Description:
;       Open a DSP parameters file in XDR format and return a product
;       handle which will be used in the subsquent calls.
;       The logical file unit is allocated by this procedure and will be
;       stored in the returned product handle.
;
;   Synopsis:
;       DSPPARM_OPEN, file_name, prod_handle
;
;   Parameters:
;       file_name               - in    - pathname of Intermediate product
;                                               or detection data file
;       prod_handle             -   out - product handle structure
;
PRO DSPPARM_OPEN, file_name, prod_handle
   hdl={t_dspparm_prod_handle}

;
; store the file name into the product handle
;
   hdl.file_name=file_name
   hdl.unit=-1  ; indicates the file has not been opened
   ON_IOERROR, bail_out
;
; open file with XDR keyword
; use the GET_LUN keyword to allocate a logical file unit
;
   OPENR, unit, file_name, /XDR, /GET_LUN

; store the logical file unit into the product handle
   hdl.unit=unit
   GOTO,  exit
;
; in case of opening error, it will print the error message.
;
bail_out:
   PRINT, !ERR_STRING

exit:
   prod_handle=hdl
;   PRINT, 'file name: ', prod_handle.file_name
;   PRINT, 'logical file unit: ', prod_handle.unit
   RETURN
END     ; end of DSPPARM_OPEN

; ==========================================================================
; PROCEDURE:    DSPPARM_CLOSE
;
;   Description:
;       close a DSP parameters file.
;
;   Synopsis:
;       DSPPARM_CLOSE, prod_handle
;
;   Parameters:
;       prod_handle             - in    - product handle structure returned
;                                         by DSPPARM_OPEN.
;
PRO DSPPARM_CLOSE, prod_handle
   IF prod_handle.unit GT 0 THEN BEGIN
      ; close and free the I/O unit
      FREE_LUN, prod_handle.unit
   END
   prod_handle.unit=-1  ; indicates the file has been closed
END     ; end of DSPPARM_CLOSE

; ==========================================================================
;
;       R E A D   F U N C T I O N S
;
; ==========================================================================
; FUNCTION:    DSPPARM_RD
;
;   Description:
;       This function reads DSP parameters from an XDR file into memory.
;       It returns a DSP processing paramters structure.
;
;   Synopsis:
;       dsp_parm = DSPPARM_RD (prod_handle)
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                               DSPPARM_OPEN
;        dsp_parm                   -   out - DSP parameters structure
;
;    Note:
;        The DSP parameters structure is a tree structure, which contain
;        parameters for the whole run, or parameters changed every DI
;        interval, NCI interval, or CPI interval.
;        All interval instances are variable-length array as defined in
;        cdsi_intf.x.   For example, the number of CPIs is varied from
;        NCI to NCI.
;        Since PV-Wave does not support variable-length array structure,
;        all instances of intervals (DI, NCI and CPI) will be assembled
;        into a fixed-length array structure with a maximum size
;        at each interval level; i.e. each NCI has 64 CPIs, each DI has
;        12 NCIs, and each run has 64 DIs maximum.
;
FUNCTION DSPPARM_RD, prod_handle
;
; declare a variable of DSP processing paramter structure
;
   DSP_parms={t_cdsi_DSP_process_parms}


;
; use SIZE function to compute the string size and call DSPPARM_RD_STRING
; to read the all pathnames and run identifier.
;
   sz = SIZE(DSP_parms.SRSD_pathname)
   str_len = sz(1)
   DSP_parms.SRSD_pathname=DSPPARM_RD_STRING(prod_handle, str_len)
;   PRINT, "SRSD_pathname: ", string(DSP_parms.SRSD_pathname(*))

   sz = SIZE(DSP_parms.RSSproc_pathname)
   str_len = sz(1)
   DSP_parms.RSSproc_pathname=DSPPARM_RD_STRING(prod_handle, str_len)
;   PRINT, "RSSproc_pathname: ", string(DSP_parms.RSSproc_pathname(*))

   sz = SIZE(DSP_parms.simdef_pathname)
   str_len = sz(1)
   DSP_parms.simdef_pathname=DSPPARM_RD_STRING(prod_handle, str_len)
;   PRINT, "simdef_pathname: ", string(DSP_parms.simdef_pathname(*))

   sz = SIZE(DSP_parms.IR_dir)
   str_len = sz(1)
   DSP_parms.IR_dir=DSPPARM_RD_STRING(prod_handle, str_len)
;   PRINT, "IR_dir: ", string(DSP_parms.IR_dir(*))

   sz = SIZE(DSP_parms.run_ident)
   str_len = sz(1)
   DSP_parms.run_ident=DSPPARM_RD_STRING(prod_handle, str_len)
;   PRINT, "run_ident: ", string(DSP_parms.run_ident(*))

   DSP_parms.altitude=DSPPARM_RD_FLOAT(prod_handle)

   antenna={t_cdsi_antenna}
   READU, prod_handle.unit, antenna
   DSP_parms.antenna=antenna
;   PRINT, DSP_parms.antenna

   DSP_parms.sampling_rate=DSPPARM_RD_FLOAT(prod_handle)
   DSP_parms.pulse_type=DSPPARM_RD_LONG(prod_handle)

;
; read number of dwells.
; NOTE: the number of dwells defined in the DSP processing parameters
;       data file is normally smaller than the maximum number of
;       dwells (c_spd_max_DI)
;

   DI_len=DSPPARM_RD_LONG(prod_handle)
   DSP_parms.DI_len=DI_len
```

```
;   PRINT, "DI_len: ", DSP_parms.DI_len

;
; loop to read all dwells which have been defined in the XDR file
;
   FOR i=0, DI_len-1 DO BEGIN
      NCI_len=DSPPARM_RD_LONG(prod_handle)
      DSP_parms.DI_val(i).NCI_len=NCI_len
;      PRINT, ''
;      PRINT, 'DI: ', i, ' NCI_len: ', NCI_len
;
;     read all NCIs within the current dwell
;
      FOR j=0, NCI_len-1 DO BEGIN
         CPI_len=DSPPARM_RD_LONG(prod_handle)
         DSP_parms.DI_val(i).NCI_val(j).CPI_len=CPI_len
;         PRINT, ''
;        PRINT, '    NCI: ', j, ' CPI_len: ', CPI_len
;
;        read all CPIs within the current NCI
;
         CPI_elm={t_cdsi_CPI_elm}
         FOR k=0, CPI_len-1 DO BEGIN
            READU, prod_handle.unit, CPI_elm
            DSP_parms.DI_val(i).NCI_val(j).CPI_val(k)=CPI_elm
;            PRINT, 'CPI_elm: ', CPI_elm
         END

         DSP_parms.DI_val(i).NCI_val(j).inter_ph_ctr_sep=DSPPARM_RD_FLOAT(prod_handle)
         DSP_parms.DI_val(i).NCI_val(j).samples_per_PRI=DSPPARM_RD_LONG(prod_handle)
         DSP_parms.DI_val(i).NCI_val(j).num_pulses_tx=DSPPARM_RD_LONG(prod_handle)
         DSP_parms.DI_val(i).NCI_val(j).num_pulses_proc=DSPPARM_RD_LONG(prod_handle)
         DSP_parms.DI_val(i).NCI_val(j).num_bursts_integ=DSPPARM_RD_LONG(prod_handle)
         DSP_parms.DI_val(i).NCI_val(j).OS_NCI_rank=DSPPARM_RD_LONG(prod_handle)

         DSP_parms.DI_val(i).NCI_val(j).capture_NCI_enable= $
                                         DSPPARM_RD_LONG(prod_handle)
;         PRINT, 'NCI_val: '
;         PRINT, '  NCI_val.inter_ph_ctr_sep:', $
;                 DSP_parms.DI_val(i).NCI_val(j).inter_ph_ctr_sep
;         PRINT, '  NCI_val.samples_per_PRI:', $
;                 DSP_parms.DI_val(i).NCI_val(j).samples_per_PRI
;         PRINT, '  NCI_val.num_pulses_tx:', $
;                 DSP_parms.DI_val(i).NCI_val(j).num_pulses_tx
;         PRINT, '  NCI_val.num_pulses_proc:', $
;                 DSP_parms.DI_val(i).NCI_val(j).num_pulses_proc
;         PRINT, '  NCI_val.num_bursts_integ:', $
;                 DSP_parms.DI_val(i).NCI_val(j).num_bursts_integ
;         PRINT, '  NCI_val.OS_NCI_rank:', $
;                 DSP_parms.DI_val(i).NCI_val(j).OS_NCI_rank
;        PRINT, '  NCI_val.capture_NCI_enable:', $
;                 DSP_parms.DI_val(i).NCI_val(j).capture_NCI_enable
      END

      engag_geom={t_cdsi_engag_geometry}
      READU, prod_handle.unit, engag_geom
      DSP_parms.DI_val(i).engag_geom=engag_geom
;      PRINT, 'engag_geom: ' & PRINT, engag_geom

      DSP_parms.DI_val(i).noise_power_main=DSPPARM_RD_FLOAT(prod_handle)
      DSP_parms.DI_val(i).noise_power_aux=DSPPARM_RD_FLOAT(prod_handle)
;      PRINT, 'noise_power_main: ', DSP_parms.DI_val(i).noise_power_main
;      PRINT, 'noise_power_aux: ',  DSP_parms.DI_val(i).noise_power_aux

      waveform={t_cdsi_DI_waveform}
      READU, prod_handle.unit, waveform
      DSP_parms.DI_val(i).waveform=waveform
;      PRINT, 'waveform: '
;      PRINT, '  waveform.replica_len: ', waveform.replica_len
;      PRINT, '  waveform.exp_pulse_dur: ', waveform.exp_pulse_dur
;       PRINT, '  waveform.compr_pulse_width: ', waveform.compr_pulse_width

      PD={t_cdsi_pulse_doppler}
      READU, prod_handle.unit, PD
      DSP_parms.DI_val(i).PD=PD
;       PRINT, 'PD: '& PRINT, PD

      ASAR_DPCA={t_cdsi_ASAR_DPCA}
      READU, prod_handle.unit, ASAR_DPCA
      DSP_parms.DI_val(i).ASAR_DPCA=ASAR_DPCA
;       PRINT, 'ASAR_DPCA: ' & PRINT, ASAR_DPCA

      detect_thrshd={t_cdsi_detect_thrshd}
      READU, prod_handle.unit, detect_thrshd
      DSP_parms.DI_val(i).detect_thrshd=detect_thrshd
;       PRINT, 'detect_thrshd: ' & PRINT, detect_thrshd

      parm_est={t_cdsi_parm_est}
      READU, prod_handle.unit, parm_est
      DSP_parms.DI_val(i).parm_est=parm_est
;       PRINT, 'parm_est: ' & PRINT, parm_est

      DSP_parms.DI_val(i).capture_DI_enable=DSPPARM_RD_LONG(prod_handle)
   END

;
; read parameters defined at the CTI level (run level)
;
   general_config={t_cdsi_general_config}
   READU, prod_handle.unit, general_config
   DSP_parms.general_config=general_config
;    PRINT, 'general_config: ', general_config

   ECCM_config={t_cdsi_ECCM_config}
   READU, prod_handle.unit, ECCM_config
   DSP_parms.ECCM_config=ECCM_config
;    PRINT, 'ECCM_config: ', ECCM_config

   ECCM_proc={t_cdsi_ECCM_proc}
   READU, prod_handle.unit, ECCM_proc
   DSP_parms.ECCM_proc=ECCM_proc
;    PRINT, 'ECCM_proc: ', ECCM_proc

   radar_proc={t_cdsi_radar_proc}
   READU, prod_handle.unit, radar_proc
   DSP_parms.radar_proc=radar_proc
;    PRINT, 'radar_proc: ', radar_proc

   IR={t_cdsi_IR_capture}
   READU, prod_handle.unit, IR
   DSP_parms.IR=IR
;    PRINT, 'IR_capture: ', IR

   sig_reduct={t_cdsi_signal_reduction}
   READU, prod_handle.unit, sig_reduct
   DSP_parms.sig_reduct=sig_reduct
;    PRINT, 'signal_reduction: ', sig_reduct

   RETURN, DSP_parms
END

; ==========================================================================
; PROCEDURE:     DSPPARM_PRINT
;
;   Description:
;       This procedure to print out the contents of the DSP processing
;       parameter structure.  This is for testing purpose.
;
;   Synopsis:
;       DSPPARM_PRINT, dsp_parm
```

```
;
;   Parameters:
;       dsp_parm                - in    - DSP parameters structure, which
;                                         was returned by DSPPARM_RD function.
;
;
PRO DSPPARM_PRINT, DSP_parms

    PRINT, "SRSD_pathname: ", string(DSP_parms.SRSD_pathname(*))
    PRINT, "RSSproc_pathname: ", string(DSP_parms.RSSproc_pathname(*))
    PRINT, "run_ident: ", string(DSP_parms.run_ident(*))
    PRINT, 'DSP_parms.altitude: ', DSP_parms.altitude
    PRINT, 'DSP_parms.antenna: ', DSP_parms.antenna
    PRINT, 'DSP_parms.sampling_rate: ', DSP_parms.sampling_rate
    PRINT, 'DSP_parms.pulse_type: ', DSP_parms.pulse_type

    DI_len = DSP_parms.DI_len
    PRINT, "DI_len: ", DSP_parms.DI_len

    FOR i=0, DI_len-1 DO BEGIN
       NCI_len = DSP_parms.DI_val(i).NCI_len
       PRINT, ''
       PRINT, 'DI: ', i, ' NCI_len: ', NCI_len
;
;      read all NCIs within the current dwell
;
       FOR j=0, NCI_len-1 DO BEGIN
          CPI_len = DSP_parms.DI_val(i).NCI_val(j).CPI_len
          PRINT, ''
          PRINT, '    NCI: ', j, ' CPI_len: ', CPI_len
;
;         read all CPIs within the current NCI
;
          FOR k=0, CPI_len-1 DO BEGIN
             PRINT, 'CPI_elm: ', DSP_parms.DI_val(i).NCI_val(j).CPI_val(k)
          END

          PRINT, 'NCI_val: '
          PRINT, '  NCI_val.inter_ph_ctr_sep:', $
                  DSP_parms.DI_val(i).NCI_val(j).inter_ph_ctr_sep
          PRINT, '  NCI_val.samples_per_PRI:', $
                  DSP_parms.DI_val(i).NCI_val(j).samples_per_PRI
          PRINT, '  NCI_val.num_pulses_tx:', $
                  DSP_parms.DI_val(i).NCI_val(j).num_pulses_tx
          PRINT, '  NCI_val.num_pulses_proc:', $
                  DSP_parms.DI_val(i).NCI_val(j).num_pulses_proc
          PRINT, '  NCI_val.num_bursts_integ:', $
                  DSP_parms.DI_val(i).NCI_val(j).num_bursts_integ
          PRINT, '  NCI_val.OS_NCI_rank:', $
                  DSP_parms.DI_val(i).NCI_val(j).OS_NCI_rank
          PRINT, '  NCI_val.capture_NCI_enable:', $
                  DSP_parms.DI_val(i).NCI_val(j).capture_NCI_enable
       END

       PRINT, 'engag_geom: ' & PRINT, DSP_parms.DI_val(i).engag_geom
       PRINT, 'noise_power_main: ', DSP_parms.DI_val(i).noise_power_main
       PRINT, 'noise_power_aux: ',  DSP_parms.DI_val(i).noise_power_aux
       PRINT, 'waveform: '
       PRINT, '  waveform.replica_len: ',  $
              DSP_parms.DI_val(i).waveform.replica_len
       PRINT, '  waveform.exp_pulse_dur: ', $
              DSP_parms.DI_val(i).waveform.exp_pulse_dur
       PRINT, '  waveform.compr_pulse_width: ', $
              DSP_parms.DI_val(i).waveform.compr_pulse_width
       PRINT, 'PD: '& PRINT, DSP_parms.DI_val(i).PD
       PRINT, 'ASAR_DPCA: ' & PRINT, DSP_parms.DI_val(i).ASAR_DPCA
       PRINT, 'detect_thrshd: ' & PRINT, DSP_parms.DI_val(i).detect_thrshd
       PRINT, 'parm_est: ' & PRINT, DSP_parms.DI_val(i).parm_est
       PRINT, 'capture_DI_enable: ', DSP_parms.DI_val(i).capture_DI_enable

    END

    PRINT, 'general_config: ', DSP_parms.general_config
    PRINT, 'ECCM_config: ', DSP_parms.ECCM_config
    PRINT, 'ECCM_proc: ', DSP_parms.ECCM_proc
    PRINT, 'radar_proc: ', DSP_parms.radar_proc
    PRINT, 'IR_capture: ', DSP_parms.IR
    PRINT, 'signal_reduction: ', DSP_parms.sig_reduct

END

```

```
; ==========================================================================
;
; SPD - Signal Processing Demonstrator Computer System
;
; (C) Copyright Her Majesty the Queen in Right of Canada
; as represented by the Minister of National Defence
;
; File Name:    cdat_dspir_intf.pro - DSP Intermediate Results include file
;
; Identification:       [@(#)]
;
; SCCS Id:      @(#)cdat_dspir_intf.pro 1.3 93/09/17
;
; Responsible Organization:
;
;               MacDonald Dettwiler and Associates
;
; Purpose:      This file declares constants and data structures for
;               ingesting DSP Intermediate Products and Detections file
;               in PV_Wave language.
;
;               The Intermediate Products and Detections data are
;               stored in XDR format.  A product file will contain
;               a header block, a control block and a data block
;               (See CSIP package). This package is depending on csip_intf.x.
;
;               Since cdsi_intf.x is depending on spd_intf.x, before loading
;               this include file into PV-Wave cdat_dspir_intf.pro (a PV-Wave
;               version of spd_intf.x) must be loaded first.
;
; Designer:     Stephen Tran
;
; Change History:
;
;    SCR  Date         Author   Change made
;    ---  -----------  ------   -------------------------------------
;    n/a  25-May-1993  cst      Initial Release
;
; ==========================================================================
; INCLUDE FILE cdat_dspir_intf.pro
;
;   Description:
;       This file contains constants and data structures defined in
;       PV-Wave language.
;       These constants and data structures were originally declared
;       in CSIP package (see csip_intf.x).
;       This includes constants, data types, Intermediate Product's header
;       block, control block and detection data structures.
;
;   Synopsis:
;       @cdat_dspir_intf
;
;       NOTE: this file must be included in the main program, before
;             calling DSP Intermediate Products ingest function
;             provided by this package.
;
; ==========================================================================
;              I M P L E M E N T A T I O N
; ==========================================================================
;
; Define constants
;
; constants in enum type: t_csip_DSP_IR_types
  c_csip_IR_ESP_input_signals = 0
  c_csip_IR_post_SC_signals = 1
  c_csip_IR_post_nulling_signals = 2
  c_csip_IR_post_blanking_signals = 3
  c_csip_IR_post_MASR_signals = 4
  c_csip_IR_post_FA_signals = 5
  c_csip_IR_post_PD_signals = 6
```

```
   c_csip_IR_DSP_input_signals = 7
   c_csip_IR_post_PC_signals = 8
   c_csip_IR_post_CC_signals = 9
   c_csip_IR_RCS_signals = 10
   c_csip_IR_blanked_signals = 11
   c_csip_IR_post_NCI_signals = 12
   c_csip_IR_post_CFAR_data = 13
   c_csip_IR_monopulse_angle = 14
   c_csip_IR_nulling_weight_vectors = 15
   c_csip_IR_correlation_matrices = 16
   c_csip_IR_cholesky_factor_matrices = 17
   c_csip_IR_adaptive_nulling_snapshots = 18
   c_csip_IR_PD_window = 19
   c_csip_IR_DPCA_weights = 20
   c_csip_IR_target_deweighting = 21
   c_csip_IR_PC_replicas = 22
   c_csip_IR_detections = 23

; constants in enum type: t_csip_update_interval
   c_csip_DI = 0
   c_csip_NCI = 1
   c_csip_CPI = 2

; constants in enum type: t_csip_element_type
   c_csip_float_elm = 0
   c_csip_double_elm = 1
   c_csip_complexS_elm = 2
   c_csip_complexD_elm = 3
   c_csip_detect_struct = 4


;
;  Data Structure of one detection data
;
TMP={t_csip_detect_elm, $
            ambig_range:0L, velocity_bin:0L, monopulse_angle:0.0, $
            integ_target_power:FLTARR(3)}
;
;  Data Structure for Header block
;
TMP={t_csip_header_block, $
           IR_id:0L, $
           SRSD_pathname:BYTARR(256), $
           RSSproc_pathname:BYTARR(256), $
           simdef_pathname:BYTARR(256), $
           interval_type:0L, dim_count:0L, elm_type:0L, elm_size:0L, $
           control_block_offset:0L, data_block_offset:0L}
;
;  Data Structure for Control block of DI update interval
;
TMP={t_csip_DI_elm, $
            DI_id:0L, dimX:0L, dimY:0L, dimZ:0L, offset:0L}
TMP={t_csip_DI_cntr_blk, DI_len:0L, $
                DI_val:REPLICATE({t_csip_DI_elm}, c_spd_max_DI) }
;
;  Data Structure for Control block of NCI update interval
;
TMP={t_csip_NCI_elm, $
             NCI_id:0L, dimX:0L, dimY:0L, dimZ:0L, offset:0L}
TMP={t_csip_NCI_cntr_elm, $
                 DI_id:0L, NCI_len:0L,$
                 NCI_val:REPLICATE({t_csip_NCI_elm}, c_spd_max_NCI) }
TMP={t_csip_NCI_cntr_blk, DI_len:0L, $
                  DI_val:REPLICATE({t_csip_NCI_cntr_elm}, c_spd_max_DI) }
;
;  Data Structure for Control block of CPI update interval
;
TMP={t_csip_CPI_elm, $
            CPI_id:0L, dimX:0L, dimY:0L, dimZ:0L, offset:0L}
TMP={t_csip_CPI_cntr_elmx, $
```

```
                  NCI_id:0L, CPI_len:0L, $
                  CPI_val:REPLICATE({t_csip_CPI_elm}, c_spd_max_CPI) }
TMP={t_csip_CPI_cntr_elm, $
                  DI_id:0L, NCI_len:0L,$
                  NCI_val:REPLICATE({t_csip_CPI_cntr_elmx}, c_spd_max_NCI) }
TMP={t_csip_CPI_cntr_blk, DI_len:0L, $
                   DI_val:REPLICATE({t_csip_CPI_cntr_elm}, c_spd_max_DI) }
;
;  Product handle data structure
;  This structure contains the unit number of Intermediate Product
;  or Detection file, header block info, and control block info.
;  The control block data structure is varied depending on
;  the update interval type: DI interval, NCI interval or CPI interval.
;  Therefore, there will be three product handle data structures, one
;  for each update interval type.
;
TMP={t_dspir_prod_handle, $
          file_name:"", $
          unit:0L }

TMP={t_dspir_prod_handle_DI, $
                file_name:"", $
                unit:0L, $
                header: {t_csip_header_block}, $
                cntr_blk: {t_csip_DI_cntr_blk}}

TMP={t_dspir_prod_handle_NCI, $
                file_name:"", $
                unit:0L, $
                header: {t_csip_header_block}, $
                cntr_blk: {t_csip_NCI_cntr_blk}}

TMP={t_dspir_prod_handle_CPI, $
                file_name:"", $
                unit:0L, $
                header: {t_csip_header_block}, $
                cntr_blk: {t_csip_CPI_cntr_blk}}

; release data memory
TMP=0L


```

```
; ==========================================================================
;
; SPD - Signal Processing Demonstrator Computer System
;
; (C) Copyright Her Majesty the Queen in Right of Canada
; as represented by the Minister of National Defence
;
; File Name:    cdat_dspir_rd.pro - Read DSP (ESP & RSP) Intermediate Products
;                                   and Detection data in PV-Wave
;
; Identification:       [@(#)]
;
; SCCS Id:      @(#)cdat_dspir_rd.pro   1.5 93/09/29
;
; Responsible Organization:
;
;               MacDonald Dettwiler and Associates
;
; Purpose:      This file implements ingest functions in PV-Wave command
;               language to read DSP Intermediate Product and Detection data
;               file in XDR format.
;
;               These files are written out using the file format described
;               in the CSIP package.  Both Intermediate Product and Detection
;               data file will contain a header block, a control block and
;               the data block including multiple instances of update
;               intervals.
;
; Designer:     Stephen Tran
;
; Change History:
;
;    SCR  Date         Author    Change made
;    ---  -----------  ------    ----------------------------------------
;    n/a 07-May-1993   cst      Initial Release
;    n/a 10-Sep-1993   cst      Implementation
;
; ========================================================================
;
; ========================================================================
;
;     R E A D    U T I L I T Y    F U N C T I O N S
;
; ========================================================================
; FUNCTION:    DSPIR_RD_STRING
;
;
;   Description:
;       This function reads a string from an XDR file. The string size in
;       characters is passed to this function.  The string size can
;       be known by applying the "SIZE" function to the string.
;       The returned string is defined as a byte array with the specified
;       string size.
;
;   Synopsis:
;       string = DSPIR_RD_STRING (prod_handle, str_len)
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         DSPPARM_OPEN
;       str_len                 - in    - string size in characters
;       string                  -   out - a string defined in byte array
;
FUNCTION DSPIR_RD_STRING, prod_handle, str_len
   str=BYTARR(str_len)
   char=0L
   FOR i=0, str_len-1 DO BEGIN
      READU, prod_handle.unit, char
      str(i)=BYTE(char,3)
   END
   RETURN, str
```

```
END     ; end of DSPIR_RD_STRING

; ===========================================================================
; FUNCTION:    DSPIR_RD_FLOAT
;
;   Description:
;       This function reads a single-precision floating point value
;       from an XDR file.
;
;   Synopsis:
;       float_val = DSPIR_RD_FLOAT (prod_handle)
;
;   Parameters:
;       prod_handle              - in    - product handle returned from
;                                          DSPPARM_OPEN
;       float_val                -   out - floating pointing value
;
FUNCTION DSPIR_RD_FLOAT, prod_handle
   float_val=0.0
   READU, prod_handle.unit, float_val
   RETURN, float_val
END     ; end of DSPIR_RD_FLOAT

; ===========================================================================
; FUNCTION:    DSPIR_RD_LONG
;
;   Description:
;       This function reads a longword integer (32 bytes)
;       from an XDR file.
;
;   Synopsis:
;       lonint_val = DSPIR_RD_LONG (prod_handle)
;
;   Parameters:
;       prod_handle              - in    - product handle returned from
;                                          DSPPARM_OPEN
;       lonint_val               -   out - longword integer value
;
FUNCTION DSPIR_RD_LONG, prod_handle
   long_val=0L
   READU, prod_handle.unit, long_val
   RETURN, long_val
END     ; end of DSPIR_RD_LONG

; ===========================================================================
; FUNCTION:    DSPIR_ASSOC
;
;   Description:
;       This function associates the element array using the provided
;       file offset and dimensions.  The individual element type is
;       specified by elm_type (t_csip_element_type): float, double,
;       complex or structure.  This function will read a data record
;       (an update interval) into memory and the data array will be
;       returned.
;
;   Synopsis:
;       data_record = DSPIR_ASSOC (prod_handle, $
;                                  offset, dimX, dimY, dimZ)
;
;
;   Parameters:
;       prod_handle              - in    - product handle returned from
;                                          DSPPARM_OPEN
;       offset                   - in    - offset of data record in bytes
;                                          from the beginning of file
;       dimX                     - in    - dimension size in X direction
;       dimY                     - in    - dimension size in Y direction
;       dimZ                     - in    - dimension size in Z direction
;       data_record              -   out - a data record  (an array in the
;                                          appropriate data type)
;
;
```

```
;
FUNCTION DSPIR_ASSOC, prod_handle, offset, dimX, dimY, dimZ

   data_record = -1
   unit = prod_handle.unit
   dim_count = prod_handle.header.dim_count
   elm_type = prod_handle.header.elm_type
;
; check and make sure offset and all dimensions are right
;
   IF (offset LT prod_handle.header.data_block_offset) OR $
      (dim_count LT 1) OR (dim_count GT 4) OR $
      (dimX LE 0) OR (dimY LE 0) OR (dimZ LE 0) THEN  GOTO, return

;
; associate the element array using the provided offset.
; the dimension of the array is defined by dimX, dimY and dimZ.
;
   CASE elm_type OF

   ; c_csip_float_elm
   0: BEGIN
         CASE dim_count OF
         1: record = ASSOC (unit, FLTARR(dimX, /Nozero), offset)
         2: record = ASSOC (unit, FLTARR(dimX, dimY, /Nozero), offset)
         3: record = ASSOC (unit, FLTARR(dimX, dimY, dimZ, /Nozero), offset)
         ENDCASE
      END

   ; c_csip_double_elm
   1: BEGIN
         CASE dim_count OF
         1: record = ASSOC (unit, DBLARR(dimX, /Nozero), offset)
         2: record = ASSOC (unit, DBLARR(dimX, dimY, /Nozero), offset)
         3: record = ASSOC (unit, DBLARR(dimX, dimY, dimZ, /Nozero), offset)
         ENDCASE
      END

   ; c_csip_complexS_elm
   2: BEGIN
         CASE dim_count OF
         1: record = ASSOC (unit, COMPLEXARR(dimX, /Nozero), offset)
         2: record = ASSOC (unit, COMPLEXARR(dimX, dimY, /Nozero), offset)
         3: record = ASSOC (unit, COMPLEXARR(dimX, dimY, dimZ, /Nozero), offset)
         ENDCASE
      END

   ; c_csip_complexD_elm
   3: BEGIN
         CASE dim_count OF
         1: record = ASSOC (unit, $
                            REPLICATE({t_spd_complexD}, dimX), $
                            offset)
         2: record = ASSOC (unit, $
                            REPLICATE({t_spd_complexD}, dimX, dimY), $
                            offset)
         3: record = ASSOC (unit, $
                            REPLICATE({t_spd_complexD}, dimX, dimY, dimZ), $
                            offset)
         ENDCASE
      END


   ; c_csip_detect_struct
   ; In the normal case it must be one-dimensional array.
   4: BEGIN
         CASE dim_count OF
         1: record = ASSOC (unit, $
                            REPLICATE({t_csip_detect_elm}, dimX), $
                            offset)
         2: record = ASSOC (unit, $
                           REPLICATE({t_csip_detect_elm}, dimX, dimY), $
                           offset)
         3: record = ASSOC (unit, $
                           REPLICATE({t_csip_detect_elm}, dimX, dimY, dimZ), $
                           offset)
         ENDCASE
      END

   ELSE:  BEGIN
          PRINT, "invalid element type."
          GOTO, return
          END

   ENDCASE
;
;
; read data record into memory
   data_record = record (0)
;
; exit
;
return:
   RETURN, data_record
END     ; end of DSPIR_ASSOC

; ========================================================================
;
;     O P E N / C L O S E    F U N C T I O N S
;
; ========================================================================
; PROCEDURE:    DSPIR_OPEN
;
;   Description:
;       Open a DSP Intermediate Product or Detections Product file
;       in XDR format.
;       The logical file unit is allocated by this procedure and will be
;       stored in the returned product handle.
;
;       This procedure reads the header block and the control block,
;       and then stores them into a product handle structure,
;       which is defined in cdat_intf.pro.  This product handle is
;       then passed to the subsquent calls for reading individual Data
;       Update Intervals.
;
;   Synopsis:
;       DSPIR_OPEN, file_name, prod_handle
;
;   Parameters:
;       file_name                - in    - pathname of Intermediate product
;                                          or detection data file
;       prod_handle              -   out - product handle structure which
;                                          contains both the header block and
;                                          control block information.
;
 PRO DSPIR_OPEN, file_name, prod_handle
    hdl={t_dspir_prod_handle}

 ;
 ; store the file name into the product handle
 ;
    hdl.file_name=file_name
    hdl.unit=-1  ; indicates the file has not been opened
    ON_IOERROR, bail_out
 ;
 ; open file with XDR keyword
 ; use the GET_LUN keyword to allocate a logical file unit
 ;
    OPENR, unit, file_name, /XDR, /GET_LUN

 ; store the logical file unit into the product handle
```

```
    hdl.unit=unit
    GOTO,  ok
 ;
 ; in case of opening error, it will print the error message.
 ;
 bail_out:
    PRINT, !ERR_STRING
    prod_handle=hdl
    RETURN


 ok:
    header_blk={t_csip_header_block}
 ;
 ;   read header block into memory and store it into product handle
 ;
    header_blk.IR_id = DSPIR_RD_LONG(hdl)
    sz = SIZE(header_blk.SRSD_pathname)
    header_blk.SRSD_pathname=DSPIR_RD_STRING(hdl, sz(1))
    sz = SIZE(header_blk.RSSproc_pathname)
    header_blk.RSSproc_pathname=DSPIR_RD_STRING(hdl, sz(1))
    sz = SIZE(header_blk.simdef_pathname)
    header_blk.simdef_pathname=DSPIR_RD_STRING(hdl, sz(1))

    header_blk.interval_type = DSPIR_RD_LONG(hdl)
    header_blk.dim_count = DSPIR_RD_LONG(hdl)
    header_blk.elm_type = DSPIR_RD_LONG(hdl)
    header_blk.elm_size = DSPIR_RD_LONG(hdl)
    header_blk.control_block_offset = DSPIR_RD_LONG(hdl)
    header_blk.data_block_offset = DSPIR_RD_LONG(hdl)

 ;
 ;  read control block into memory and store it into product handle
 ;
 ;   PRINT, "control_block:"
    CASE header_blk.interval_type OF

 ; case interval_type = c_csip_DI
      0: BEGIN

 ; prod_handle is declared as t_dspir_prod_handle_DI type

       prod_handle={t_dspir_prod_handle_DI}
       prod_handle.file_name = hdl.file_name
       prod_handle.unit = hdl.unit
       prod_handle.header = header_blk

       len_pntr = ASSOC(unit, LONARR(1), header_blk.control_block_offset)
       DI_len = len_pntr(0)
       prod_handle.cntr_blk.DI_len = DI_len(0)

 ; Read actual number of DI elements into an array
       cntr_block_pntr = ASSOC(unit, $
                                   {t_csip_DI_elm},  $
                                   header_blk.control_block_offset + 4)


       FOR i=0, prod_handle.cntr_blk.DI_len-1 DO BEGIN
          prod_handle.cntr_blk.DI_val(i) = cntr_block_pntr(i)
       ENDFOR
   END ;c_csip_DI


; case interval_type = c_csip_NCI
   1: BEGIN

; prod_handle is declared as the type of t_dspir_prod_handle
      prod_handle={t_dspir_prod_handle_NCI}
      prod_handle.file_name = hdl.file_name
      prod_handle.unit = hdl.unit
      prod_handle.header = header_blk
      len_pntr = ASSOC(unit, LONARR(1), header_blk.control_block_offset)
      DI_len = len_pntr(0)
      prod_handle.cntr_blk.DI_len = DI_len(0)

      NCI_val = {t_csip_NCI_elm}
;     loop for all dwells
      FOR i=0, prod_handle.cntr_blk.DI_len-1 DO BEGIN
         prod_handle.cntr_blk.DI_val(i).DI_id = DSPIR_RD_LONG(hdl)
         prod_handle.cntr_blk.DI_val(i).NCI_len = DSPIR_RD_LONG(hdl)
;        loop for all NCIs
         FOR j=0, prod_handle.cntr_blk.DI_val(i).NCI_len-1 DO BEGIN
            READU, unit, NCI_val
            prod_handle.cntr_blk.DI_val(i).NCI_val(j) = NCI_val
         ENDFOR
      ENDFOR
   END ;c_csip_NCI

; case interval_type = c_csip_CPI
  2: BEGIN

; prod_handle is declared as the type of t_dspir_prod_handle_CPI
      prod_handle={t_dspir_prod_handle_CPI}
      prod_handle.file_name = hdl.file_name
      prod_handle.unit = hdl.unit
      prod_handle.header = header_blk
      len_pntr = ASSOC(unit, LONARR(1), header_blk.control_block_offset)

      DI_len = len_pntr(0)
      prod_handle.cntr_blk.DI_len = DI_len(0)

      NCI_val = {t_csip_CPI_cntr_elmx}
      CPI_val = {t_csip_CPI_elm}
;     loop for all dwells
      FOR i=0, prod_handle.cntr_blk.DI_len-1 DO BEGIN

         prod_handle.cntr_blk.DI_val(i).DI_id = DSPIR_RD_LONG(hdl)
         prod_handle.cntr_blk.DI_val(i).NCI_len = DSPIR_RD_LONG(hdl)

;        loop for all NCIs
         FOR j=0, prod_handle.cntr_blk.DI_val(i).NCI_len-1 DO BEGIN
            prod_handle.cntr_blk.DI_val(i).NCI_val(j).NCI_id  = $
                 DSPIR_RD_LONG(hdl)
            prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_len = $
                 DSPIR_RD_LONG(hdl)

;           loop for all CPIs
            FOR k=0, prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_len-1 $
            DO BEGIN
               READU, unit, CPI_val
               prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_val(k) = CPI_val
            ENDFOR
         ENDFOR
      ENDFOR
   END ;c_csip_CPI

; other cases
   ELSE: BEGIN
         PRINT, "the input file has invalid interval type."
         END

   ENDCASE

   RETURN
END     ; end of DSPIR_OPEN



; ==========================================================================
; PROCEDURE:    DSPIR_CLOSE
```

```
;
;   Description:
;       close a DSP Intermediate Product or Detections Product file.
;
;   Synopsis:
;       DSPIR_CLOSE, prod_handle
;
;   Parameters:
;       prod_handle             - in    - product handle structure returned
;                                         by DSPIR_OPEN.
PRO DSPIR_CLOSE, prod_handle
   IF prod_handle.unit GT 0 THEN BEGIN
      ; close and free the I/O unit
      FREE_LUN, prod_handle.unit
   END
   prod_handle.unit=-1  ; indicates the file has been closed
END     ; end of DSPIR_CLOSE


; ======================================================================
;
;     R E A D   F U N C T I O N S
;
; ======================================================================
; FUNCTION:    DSPIR_RD_DI
;
;   Description:
;       Given a Dwell Identifier, this function reads the selected Dwell
;       Data Update Interval from the product file into memory.
;       Remember that the file offset to the individual Data Update Interval
;       is stored in the control block.  Once the header block and control
;       block of the product file is read, any Data Update Interval
;       can then be accessed randomly from the file.
;
;   Synopsis:
;       di_data = DSPIR_RD_DI (prod_handle, DI_id)
;
;   Exceptions:
;       If prod_handle indicates that this product is a NCI Interval or a CPI
;       Interval type, this function will report errors and no Data Update
;       Interval will be read.
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         DSPIR_OPEN
;       DI_id                   - in    - Dwell Identifier
;       di_data                 -   out - a DI Data Update Interval.
;                                         An array structure is returned.
;                                         If error is detected or no data is
;                                         found in the file (missing or empty),
;                                         -1 is returned.
;
;                                         This array is declared and memory
;                                         is allocated as a 1D, 2D or 3D
;                                         matrix of either floating-point,
;                                         complex or detection structure
;                                         data elements.
;                                         The element data type is varied
;                                         depending on the product type.
;                                         In addition, the dimensions
;                                         (dimX, dimY, dimZ) used to declare
;                                         this array are stored in the control
;                                         block and were read in by DSPIR_OPEN.
;
;
;   Note:
;       This function does NOT return the size and element data type of
;       "di_data" to the caller; however, these information can be obtained
;       by calling either DSPIR_SIZE function provided in this package or
;       PV-Wave SIZE function.
```

```
;
FUNCTION DSPIR_RD_DI, prod_handle, DI_id

   di_data = -1
   CASE prod_handle.header.interval_type OF

   0: BEGIN
         FOR i=0, prod_handle.cntr_blk.DI_len-1 DO BEGIN
            IF DI_id EQ prod_handle.cntr_blk.DI_val(i).DI_id THEN BEGIN

               offset = prod_handle.cntr_blk.DI_val(i).offset
               dimX = prod_handle.cntr_blk.DI_val(i).dimX
               dimY = prod_handle.cntr_blk.DI_val(i).dimY
               dimZ = prod_handle.cntr_blk.DI_val(i).dimZ
               di_data = DSPIR_ASSOC (prod_handle, offset, dimX, dimY, dimZ)

               GOTO, return
             END
          ENDFOR

          ; the specified DI_id is not found in the control block

          PRINT, "the specified DI_id: ", DI_id, $
                 " is not captured in the product file (", $
                 prod_handle.file_name, ")"
       END

   ELSE: BEGIN
         PRINT, "the product file (", prod_handle.file_name, ")", $
                " is not a product of DI interval type"
         END
   ENDCASE

return:
   RETURN, di_data
END     ; end of DSPIR_RD_DI

; =====================================================================
; FUNCTION:    DSPIR_RD_NCI
;
;   Description:
;       Given a Dwell Identifier and a NCI Identifier, this function reads
;       the selected NCI Data Update Interval from the product file into memory.
;       Remember that the file offset to the individual Data Update Interval
;       is stored in the control block.  Once the header block and control
;       block of the product file is read, any Data Update Interval
;       can then be accessed randomly from the file.
;
;       Note: This function will read a detection vector (1D array of detection
;             structure)
;
;
;   Synopsis:
;       nci_data = DSPIR_RD_NCI (prod_handle, DI_id, NCI_id)
;
;   Exceptions:
;       If prod_handle indicates that this product is a DI Interval or a CPI
;       Interval type, this function will report errors and no Data
;       Update Interval will be read.
;
;   Note:
;       This function does NOT return the size and element data type of
;       "nci_data" to the caller; however, these information can be obtained
;       by calling either DSPIR_SIZE function provided in this package or
;       PV-Wave SIZE function.
;
;   Parameters:
;       prod_handle               - in    - product handle returned from
;                                           DSPIR_OPEN
;
;       DI_id                     - in    - Dwell Identifier
;       NCI_id                  - in    - NCI Identifier
;       nci_data                -   out - a NCI Data Update Interval.
;                                         It is declared as 1D, 2D or 3D array
;                                         of either floating-point, complex or
;                                         detection data type.
;
;                                         If error is detected or no data
;                                         element is found in the file
;                                         (missing or empty), -1 is returned.
;                                         For example, when the detection data
;                                         has zero count, i.e. the specified
;                                         NCI contains no detections,
;                                         nci_data is returned with -1.
;
;
FUNCTION DSPIR_RD_NCI, prod_handle, DI_id, NCI_id

   nci_data = -1
   CASE prod_handle.header.interval_type OF

   1: BEGIN

;        loop for all dwells
         FOR i=0, prod_handle.cntr_blk.DI_len-1 DO BEGIN
            IF DI_id EQ prod_handle.cntr_blk.DI_val(i).DI_id THEN BEGIN

;              loop for all NCIs
               FOR j=0, prod_handle.cntr_blk.DI_val(i).NCI_len-1 DO BEGIN
                 IF NCI_id EQ prod_handle.cntr_blk.DI_val(i).NCI_val(j).NCI_id $
                 THEN BEGIN
                    offset = prod_handle.cntr_blk.DI_val(i).NCI_val(j).offset
                    dimX = prod_handle.cntr_blk.DI_val(i).NCI_val(j).dimX
                    dimY = prod_handle.cntr_blk.DI_val(i).NCI_val(j).dimY
                    dimZ = prod_handle.cntr_blk.DI_val(i).NCI_val(j).dimZ
                    nci_data = DSPIR_ASSOC (prod_handle, $
                                            offset, dimX, dimY, dimZ)

                    GOTO, return
                 END
               ENDFOR
            END
          ENDFOR

          ; the specified DI_id is not matched in the control block

          PRINT, "the specified DI_id: ", DI_id, $
                 " is not captured in the product file (", $
                   prod_handle.file_name, ")"
        END

    ELSE: BEGIN
          PRINT, "the product file (", prod_handle.file_name, ")", $
                 " is not a product of NCI interval type"
          END
    ENDCASE

  return:
    RETURN, nci_data
  END      ; end of DSPIR_RD_NCI

; =========================================================================
; FUNCTION:    DSPIR_RD_CPI
;
;   Description:
;       Given a Dwell Identifier, a NCI Identifier and a CPI Identifier,
;       this function reads the selected CPI data Update Interval from
;       the product file into memory.
;
;       Remember that the file offset to the individual Data Update Interval
;       is stored in the control block.  Once the header block and control
;       block of the product file is read, any Data Update Interval
;       can then be accessed randomly from the file.
```

```
;
;    Synopsis:
;        cpi_data = DSPIR_RD_CPI (prod_handle, DI_id, NCI_id, CPI_id)
;
;    Exceptions:
;        If prod_handle indicates this product is a Dwell Interval or a NCI
;        Interval, this function will report errors and no data will be
;        returned.
;
;    Note:
;        This function does NOT return the size and element data type of
;        "nci_data" to the caller; however, these information can be obtained
;        by calling either DSPIR_SIZE function provided in this package or
;        PV-Wave SIZE function.
;
;    Parameters:
;        prod_handle             - in    - product handle returned from
;                                          DSPIR_OPEN
;        DI_id                   - in    - Dwell Identifier
;        NCI_id                  - in    - NCI Identifier
;        CPI_id                  - in    - CPI Identifier
;        cpi_data                -    out - a CPI Data Update Interval.
;                                          It is declared as 1D, 2D or 3D array
;                                          of either floating-point, complex or
;                                          structure data type.
;
;                                          If error is detected or no data is
;                                          found in the file (missing or empty),
;                                          -1 is returned.
;
FUNCTION DSPIR_RD_CPI, prod_handle, DI_id, NCI_id, CPI_id

   cpi_data = -1
   CASE prod_handle.header.interval_type OF

   2: BEGIN
      FOR i=0, prod_handle.cntr_blk.DI_len-1 DO BEGIN
        IF DI_id EQ prod_handle.cntr_blk.DI_val(i).DI_id THEN BEGIN
          FOR j=0, prod_handle.cntr_blk.DI_val(i).NCI_len-1 DO BEGIN
            IF NCI_id EQ prod_handle.cntr_blk.DI_val(i).NCI_val(j).NCI_id $
            THEN BEGIN
               FOR k=0, $
               prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_len-1 $
               DO BEGIN
                 IF CPI_id EQ $
                 prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_val(k).CPI_id $
                 THEN BEGIN

                   offset = $
                     prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_val(k).offset
                   dimX = $
                     prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_val(k).dimX
                   dimY = $
                     prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_val(k).dimY
                   dimZ = $
                     prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_val(k).dimZ
                   cpi_data = DSPIR_ASSOC (prod_handle, $
                                           offset, dimX, dimY, dimZ)

                   GOTO, return
                 END
               ENDFOR ; end of checking CPI_id

            END
          ENDFOR ; end of checking NCI_id

        END
      ENDFOR ; end of checking DI_id

      ; the specified DI_id is not matched in the control block
```

```

       PRINT, "the specified DI_id: ", DI_id, $
              "NCI_id: ", NCI_id, "CPI_id: ", CPI_id, $
              " is not captured in the product file (", $
              prod_handle.file_name, ")"

      END ; case c_csip_CPI

   ELSE: BEGIN
         PRINT, "the product file (", prod_handle.file_name, ")", $
                " is not a product of CPI interval type"
         END
  ENDCASE

return:
  RETURN, cpi_data
END     ; end of DSPIR_RD_CPI

; =============================================================================
;
;     E N Q U I R Y   F U N C T I O N S
;
; =============================================================================
;
; FUNCTION:    DSPIR_PROD_TYPE
;
;   Description:
;       Return the particular type of DSP Intermediate Product given a product
;       handle.
;
;   Synopsis:
;       IR_type = DSPIR_PROD_TYPE (prod_handle)
;
;   Parameters:
;       prod_handle              - in    - product handle returned from
;                                          DSPIR_OPEN
;       IR_type                  -   out - Intermediate Product type (enum data
;                                          type)
;                                          0 - 22:  Intermediate product
;                                          23:      Detections product
;
;                                          c_csip_IR_ESP_input_signals = 0,
;                                          c_csip_IR_post_SC_signals = 1,
;                                          c_csip_IR_post_nulling_signals = 2,
;                                          c_csip_IR_post_blanking_signals = 3,
;                                          c_csip_IR_post_MASR_signals = 4,
;                                          c_csip_IR_post_FA_signals = 5,
;                                          c_csip_IR_post_PD_signals = 6,
;                                          c_csip_IR_DSP_input_signals = 7,
;                                          c_csip_IR_post_PC_signals = 8,
;                                          c_csip_IR_post_CC_signals = 9,
;                                          c_csip_IR_RCS_signals = 10,
;                                          c_csip_IR_blanked_signals = 11,
;                                          c_csip_IR_post_NCI_signals = 12,
;                                          c_csip_IR_post_CFAR_data = 13,
;                                          c_csip_IR_monopulse_angle = 14,
;                                          c_csip_IR_nulling_weight_vectors = 15,
;                                          c_csip_IR_correlation_matrices = 16,
;                                          c_csip_IR_cholesky_factor_matrices = 17,
;                                          c_csip_IR_adaptive_nulling_snapshots = 18,
;                                          c_csip_IR_PD_window = 19,
;                                          c_csip_IR_DPCA_weights = 20,
;                                          c_csip_IR_target_deweighting = 21,
;                                          c_csip_IR_PC_replicas = 22,
;                                          c_csip_IR_detections = 23
;
FUNCTION DSPIR_PROD_TYPE, prod_handle
   RETURN, prod_handle.header.IR_id
END      ; end of DSPIR_PROD_TYPE

; =============================================================================
; FUNCTION:    DSPIR_INTERV_TYPE
;
;   Description:
;       Return the particular update interval type of a product file given
;       a product handle.
;       The update interval type can be either Dwell Interval, Non-coherent
;       Interval (NCI) or Coherent Processing Interval (CPI).
;
;   Synopsis:
;       interv_type = DSPIR_INTERV_TYPE (prod_handle)
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         DSPIR_OPEN
;       interv_type             -   out - update interval type: DI, NCI or CPI
;                                         (enum type)
;                                                c_csip_DI = 0,
;                                                c_csip_NCI = 1,
;                                                c_csip_CPI = 2
;
;                                         This indicates whether the instances
;                                         of a Data Update Interval is either
;                                         a dwell interval, a NCI interval
;                                         or a CPI interval.
;
FUNCTION DSPIR_INTERV_TYPE, prod_handle
   RETURN, prod_handle.header.interval_type
END      ; end of DSPIR_INTERV_TYPE

; =========================================================================
; FUNCTION:    DSPIR_DIM_COUNT
;
;   Description:
;       Return the dimensionality of the Data Update Interval. It is an
;       integer number ranging from 1 to 3.
;
;   Synopsis:
;       dim_count = DSPIR_DIM_COUNT (prod_handle)
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         DSPIR_OPEN
;       dim_count               -   out - dimension count of the Data Update
;                                         Interval: 1, 2, or 3
FUNCTION DSPIR_DIM_COUNT, prod_handle
   RETURN, prod_handle.header.dim_count
END      ; end of DSPIR_DIM_COUNT

; =========================================================================
; PROCEDURE:    DSPIR_DI_IDARR
;
;   Description:
;       Given a product handle, this function returns the information of
;       Dwell Update Intervals captured in the product file including
;       the total number of Dwell Intervals captured and their identifiers.
;
;   Synopsis:
;       DSPIR_DI_IDARR, prod_handle, di_num, di_idarr
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         DSPIR_OPEN
;       di_num                  -   out - the number of Dwell Intervals captured
;                                         in this product file.
;       di_idarr                -   out - an array of dwell identifiers.
;                                         The dwell identifier is a sequence
;                                         number within a CTI, integer number
;                                         starting from 1.
;                                         The size of this array is di_num.
;                                         Since the Intermediate Product
;                                           may be decimated through Dwell
;                                           intervals, the Dwell Intentifiers
;                                           are not always contiguous.
;
PRO DSPIR_DI_IDARR, prod_handle, di_num, di_idarr
  di_num = prod_handle.cntr_blk.DI_len
  di_idarr = prod_handle.cntr_blk.DI_val(0:di_num-1).DI_id
  RETURN
END     ; DSPIR_DI_IDARR

; ============================================================================
; PROCEDURE:    DSPIR_NCI_IDARR
;
;   Description:
;       For the NCI interval products, given a product handle and a Dwell
;       identifier, this function returns the information of NCI Data
;       Update Intervals captured in the selected dwell including the
;       total number of NCI Intervals captured and their identifiers.
;
;   Synopsis:
;       DSPIR_NCI_IDARR, prod_handle, DI, nci_num, nci_idarr
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         DSPIR_OPEN
;       DI_id                   - in    - Dwell Identifier
;       nci_num                 -   out - the number of NCI Intervals captured
;                                         in a dwell specified by DI.
;                                         If the product interval type is
;                                         Dwell Update Interval, then zero
;                                         is returned; i.e. no NCI intervals
;                                         is captured in this product.
;       nci_idarr               -   out - an array containing the NCI Interval
;                                         Identifiers (integer starting from 1).
;                                         The size of this array is nci_num.
;                                         Since the Intermediate Product may be
;                                         decimated through NCI intervals,
;                                         the NCI Intentifiers are not always
;                                         contiguous.
;
PRO DSPIR_NCI_IDARR, prod_handle, DI_id, nci_num, nci_idarr

  nci_num = 0
  nci_idarr = -1

  IF (prod_handle.header.interval_type EQ 1) OR $
     (prod_handle.header.interval_type EQ 2) THEN BEGIN
;
;    interval type is c_csip_NCI
;
     FOR i=0, prod_handle.cntr_blk.DI_len-1 DO BEGIN
        IF DI_id EQ prod_handle.cntr_blk.DI_val(i).DI_id THEN BEGIN
;
;          declare size of nci_idarr (array of longword integer)
;
           nci_num = prod_handle.cntr_blk.DI_val(i).NCI_len
           nci_idarr = $
               prod_handle.cntr_blk.DI_val(i).NCI_val(0:nci_num-1).NCI_id

           RETURN
        END
     ENDFOR
     PRINT, "the specified DI_id: ", DI_id, $
            " is not captured in the product file (", prod_handle.file_name, $
            ")."

; other interval types

  ENDIF ELSE BEGIN
     PRINT, "this product file (", prod_handle.file_name, $
            ") does not captures NCIs"
  ENDELSE
  RETURN
END     ; end of DSPIR_NCI_IDARR

; ============================================================================
; PROCEDURE:    DSPIR_CPI_IDARR
;
;   Description:
;       For the CPI interval products, given a product handle, a Dwell
;       identifier and a NCI identifier, this function returns the
;       information of CPI Data Update Intervals captured
;       in the selected dwell and NCI.  The returned information includes the
;       total number of CPI Intervals captured and their identifiers.
;       If the product interval type is Dwell interval or NCI interval,
;       zero will be returned for the totatl number of CPI Intervals
;       captured.
;
;   Synopsis:
;       DSPIR_CPI_IDARR, prod_handle, DI_id, NCI_id, cpi_num, cpi_idarr
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         DSPIR_OPEN
;       DI_id                   - in    - Dwell Identifier
;       NCI_id                  - in    - NCI Identifier
;       cpi_num                 -   out - the total number of CPI Intervals
;                                         captured in a Non-Coheren Interval
;                                         within a Dwell Interval.  The NCI
;                                         interval and dwell interval are
;                                         specified by NCI_id and DI_id
;                                         respectively.
;                                         If the product interval type is Dwell
;                                         or NCI Update Interval, then zero is
;                                         returned; i.e. no CPI intervals in
;                                         this product.
;       cpi_idarr               -   out - an array containing the CPI Interval
;                                         Identifiers (integer number starting
;                                         from 1).
;                                         The size of this array is cpi_num.
;                                         Since the Intermediate Product may be
;                                         decimated through CPI intervals,
;                                         the CPI Intentifiers are not always
;                                         contiguous.
PRO DSPIR_CPI_IDARR, prod_handle, DI_id, NCI_id, cpi_num, cpi_idarr
  cpi_num = 0
  cpi_idarr = -1

  IF prod_handle.header.interval_type EQ 2 THEN BEGIN
;
;    interval type is c_csip_CPI
;
     FOR i=0, prod_handle.cntr_blk.DI_len-1 DO BEGIN
        IF DI_id EQ prod_handle.cntr_blk.DI_val(i).DI_id THEN BEGIN

           FOR j=0, prod_handle.cntr_blk.DI_val(i).NCI_len-1 DO BEGIN
              IF NCI_id EQ prod_handle.cntr_blk.DI_val(i).NCI_val(j).NCI_id $
              THEN BEGIN

;
;          declare size of cpi_idarr (array of longword integer)
;
                 cpi_num = $
                     prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_len
                 cpi_idarr = $
prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_val(0:cpi_num-1).CPI_id

                 GOTO, return
              END
           ENDFOR
        END
     ENDFOR
     PRINT, "the specified DI_id: ", DI_id, " and NCI_id: ", NCI_id, $
            " is not captured in the product file (", prod_handle.file_name, $
            ")."

; other interval types

  ENDIF ELSE BEGIN
     PRINT, "this product file (", prod_handle.file_name, $
            ") does not captures CPIs"
  ENDELSE

return:
  RETURN
END     ; DSPIR_CPI_IDARR

; ========================================================================
; PROCEDURE:    DSPIR_SIZE
;
;   Description:
;       Given a Data Update Interval, this procedure returns: data element
;       type and dimension size.  For invalid interval data (e.g. missing
;       or empty detections), -1 will be returned in data type, dimension
;       size.
;
;   Synopsis:
;       DSPIR_SIZE, interv_data, data_type, dim_count, dimX, dimY, dimZ
;
;   Parameters:
;       interv_data             - in    - Data Update Interval
;       data_type               -   out - data element type code.
;                                         code = -1:    unknown type
;                                         code = 0:     floating-point
;                                                       (single-precission)
;                                         code = 1:     double-precision
;                                                       floating
;                                         code = 2:     single-precision complex
;                                         code = 3:     double-precision complex
;                                         code = 4:     detection structure
;
;       dim_count               -   out - dimensionality;
;                                         integer number 1, 2, or 3
;                                         -1 for missing or empty interval data
;       dimX                    -   out - size of interv_data in X direction
;                                         -1 for missing or empty interval data
;       dimY                    -   out - size of interv_data in Y direction
;                                         -1 for missing or empty interval data
;       dimZ                    -   out - size of interv_data in Z direction
;                                         -1 for missing or empty interval data
;
PRO DSPIR_SIZE, interv_data, data_type, dim_count, dimX, dimY, dimZ

    B = SIZE(interv_data)
    data_type = -1
    dim_count = B(0)
    dimX = -1
    dimY = -1
    dimZ = -1

    CASE dim_count OF
    1: BEGIN
       dimX = B(1)
       dimY = 1
       dimZ = 1
       END
    2: BEGIN
       dimX = B(1)
       dimY = B(2)
       dimZ = 1
       END
    3: BEGIN
       dimX = B(1)
       dimY = B(2)
       dimZ = B(3)
       END
    ELSE: BEGIN
          dim_count = -1
          GOTO, return
          END
    ENDCASE

    type = B(dim_count+1)

    CASE type OF
    4:   data_type = 0  ; single-precision floating-point
    5:   data_type = 1  ; double-precision floating-point
    6:   data_type = 2  ; single-precision complex
    8: BEGIN
         data_type = 3  ; double-precision complex
;
;        if data element is double-precision complex, then it has
;        2 fields (real and imaginery) in the structure (data_type=3).
;        otherwise, it must be a detection structure (data_type = 4).
;
         IF N_TAGS(interv_data) GT 2 THEN data_type = 4 ; detection structure
       END
    ELSE:    data_type = -1  ; unknown data type
    ENDCASE

return:
RETURN
END     ; end of DSPIR_SIZE

; ===========================================================================
; PROCEDURE:    DSPIR_PRINT
;
;   Description:
;       This procedure to print out the contents of the header block
;       and the control block.  This is for testing purpose.
;
;   Synopsis:
;       DSPIR_PRINT, prod_handle
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         DSPIR_OPEN
;
PRO DSPIR_PRINT, prod_handle
   header_blk = prod_handle.header
   PRINT, "header_block:"
   PRINT, "  IR_id:", header_blk.IR_id
   PRINT, "  SRSD_pathname: ", string(header_blk.SRSD_pathname(*))
   PRINT, "  RSSproc_pathname: ", string(header_blk.RSSproc_pathname(*))
   PRINT, "  simdef_pathname: ", string(header_blk.simdef_pathname(*))
   PRINT, "  interval_type:", header_blk.interval_type
   PRINT, "  dim_count:", header_blk.dim_count
   PRINT, "  elm_type:", header_blk.elm_type
   PRINT, "  elm_size:", header_blk.elm_size
   PRINT, "  control_block_offset:", header_blk.control_block_offset
   PRINT, "  data_block_offset:", header_blk.data_block_offset

   PRINT, "control_block:"
   CASE header_blk.interval_type OF

; case interval_type = c_csip_DI
   0: BEGIN

      PRINT, prod_handle.cntr_blk.DI_len
       FOR i=0, prod_handle.cntr_blk.DI_len-1 DO BEGIN
```

```
          PRINT, prod_handle.cntr_blk.DI_val(i)
      ENDFOR
   END ;c_csip_DI


; case interval_type = c_csip_NCI
   1: BEGIN

      PRINT, "DI_len: ", prod_handle.cntr_blk.DI_len
      FOR i=0, prod_handle.cntr_blk.DI_len-1 DO BEGIN
         PRINT, "DI_id: ", prod_handle.cntr_blk.DI_val(i).DI_id
         PRINT, "NCI_len:", prod_handle.cntr_blk.DI_val(i).NCI_len
         FOR j=0, prod_handle.cntr_blk.DI_val(i).NCI_len-1 DO BEGIN
             PRINT, prod_handle.cntr_blk.DI_val(i).NCI_val(j)
         ENDFOR
      ENDFOR
   END ;c_csip_NCI

; case interval_type = c_csip_CPI
   2: BEGIN
      PRINT, "DI_len: ", prod_handle.cntr_blk.DI_len
      FOR i=0, prod_handle.cntr_blk.DI_len-1 DO BEGIN
         PRINT, "DI_id: ", prod_handle.cntr_blk.DI_val(i).DI_id
         PRINT, "NCI_len:", prod_handle.cntr_blk.DI_val(i).NCI_len

         FOR j=0, prod_handle.cntr_blk.DI_val(i).NCI_len-1 DO BEGIN
            PRINT, "NCI_id: ", prod_handle.cntr_blk.DI_val(i).NCI_val(j).NCI_id
            PRINT, "CPI_len:", prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_len

            FOR k=0, prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_len-1 $
            DO BEGIN
               PRINT, prod_handle.cntr_blk.DI_val(i).NCI_val(j).CPI_val(k)
            ENDFOR
         ENDFOR
      ENDFOR
   END ;c_csip_CPI

; other cases
   ELSE: BEGIN
         PRINT, "the input file has invalid interval type."
         END

   ENDCASE
   RETURN
END      ; end of DSPIR_PRINT


; =========================================================================
;
; EXAMPLE OF HOW TO USE SIZE FUNCTION
;
; PV-Wave provides a SIZE function which gives the type and dimesion
; information of an expression.  Therefore, an experenced programmer
; can use this function to get back the size information of a Data Update
; Interval, which is a variable-length array of elements of either float,
; complex or struture.
;
; SIZE function returns a six-element vector as described in following
; example (See PV-Wave Command Language, verson 3.1, Technical
; reference Manual):
;
;              B = SIZE(array_data)
;
;              dim_count = B(0)  :      either 1, 2 or 3
;              dimX = B(1)       :      dimension in x direction
;              dimY = B(2)       :      dimension in y direction
;              dimZ = B(3)       :      dimension in z direction
;              type = B(4)       :      type code
;
;                                       1         - Byte
;                                         2       - Integer
;                                         3       - Longword integer
;                                         4       - Floating-point
;                                         5       - Double-precision floating
;                                         6       - complex floating
;                                         7       - string
;                                         8       - structure
;
;               tot_elm = B(5)   :        total number of element
;                                         (dimX*dimY*dimZ)


; ===========================================================================
; CALLING SEQUENCE EXAMPLES:
;
; The following examples demonstrate how to use the functions provided by
; this package.
;
;@cdat_spd_intf               ; read spd include file
;@cdat_dspir_intf             ; read IR include file
;.run cdat_dspir_rd           ; compile and run cdat_dspir_rd.pro
;
;
; EXAMPLE 1: read one Data Updata Interval
;
; DSPIR_OPEN, "test.dt.detection", prod_handle
;
; DI=1  ; first DI
; NCI=2 ; second NCI
;
;    ; read detection data of DI #1, NCI #2
;
; detect_data = DSPIR_RD_NCI(prod_handle, DI, NCI)
; PRINT, SIZE(detect_data)      ; print size information
;
;
; EXAMPLE 2: read entire product file from DI to DI, from NCI to NCI, and
;            from CPI to CPI.
;
; DSPIR_OPEN, "test.ir.rsp_inp", prod_hdl       ; open a RSP input signal file
;
;
; call DSPIR_DI_IDARR to get all DI identifiers
;
; DSPIR_DI_IDARR, prod_hdl, di_num, di_idarr
;
;
; ; loop for all dwells
; FOR i=0, di_num-1 DO BEGIN
;    DI = di_idarr(i)           ; the current DI identifier
;
;    ; call DSPIR_NCI_IDARR to get all NCI identifiers in a dwell
;    DSPIR_NCI_IDARR, prod_hdl, DI, nci_num, nci_idarr
;
;    ; loop for all NCIs
;    FOR j=0, nci_num-1 DO BEGIN
;       NCI = nci_idarr(j)             ; the current NCI identifier
;
;       IR_NCI_ids, prod_hdl, DI, NCI, cpi_num, cpi_idarr
;
;       ; loop for all CPIs
;       FOR k=0, cpi_num DO BEGIN
;           CPI = cpi_idarr(k)         ; the current CPI identifier
;
;           print, ' '
;           print, 'DI#: ', DI, ',  NCI#: ', NCI, ',  CPI#: ', CPI
;
;           ; read a CPI Data Update Interval
;
;           interv = DSPIR_RD_CPI (prod_hdl, DI, NCI, CPI)
;             DSPIR_SIZE, interv, data_type, dim_count, dimX, dimY, dimZ
;             print, 'dim_count: ', dim_count, $
;                    "dimX: ", dimX, "dimY: ", dimY, "dimZ: ", dimZ
;
;             ; Use PV-Wave HELP procedure to print out dimensions of the
;             ; CPI Update Interval
;             HELP, interv
;
;             PRINT, interv
;         ENDFOR
;     ENDFOR
; ENDFOR
;
;
; DSPIR_CLOSE, prod_hdl         ; close product file
; ============================================================================
; ==========================================================================
;
; SPD - Signal Processing Demonstrator Computer System
;
; (C) Copyright Her Majesty the Queen in Right of Canada
; as represented by the Minister of National Defence
;
; File Name:    cdat_rssir_intf.pro - RSS Intermediate Results include file
;
; Identification:       [@(#)]
;
; SCCS Id:      %W% %E%
;
; Responsible Organization:
;
;               Raytheon Canada Limited
;
; Purpose:      This file declares constants and data structures for
;               ingesting RSS Intermediate Products in PV_Wave language.
;
;               The Intermediate Products data are stored in XDR format.
;               A product file will contain a header block, a control block
;               and a data block
;               (See CGEP package).
;
;               This package is depending on cgep_intf.x
;
; Designer:     Kerry Zielke
;
; Change History:
;
;   SCR  Date        Author   Change made
;   --- ----------- ------  ----------------------------------------
;   n/a 22-Oct-1993  kmz    Initial Release
;
; ==========================================================================
; INCLUDE FILE cdat_rssir_intf.pro
;
;   Description:
;       This file contains constants and data structures defined in
;       PV-Wave language.
;       These constants and data structures were originally declared
;       in the CGEP package (see cgep_intf.x).
;       These include generic constants, generic data type, Intermediate
;       Product's header block, control block and data structures.
;
;   Synopsis:
;       @cdat_rssir_intf
;
;       NOTE: this file must be included first in the main program, before
;       calling any procedure or function provided by this package.
;
;
; ==========================================================================
;                I M P L E M E N T A T I O N
; ==========================================================================
;
; Define constants
;
; constants in enum type: t_cgep_filetype
```

```
   c_cgep_rss_clut = 0
   c_cgep_rss_exn = 1
   c_cgep_rss_jam = 2
   c_cgep_rss_tar = 3
   c_cgep_rss_comp = 4
   c_cgep_rss_post_nl = 5
   c_cgep_rss_if_filt = 6
   c_cgep_rss_post_sd = 7
   c_cgep_rss_ant_patrn = 8
   c_cgep_rss_filt_rsp = 9
   c_cgep_rss_post_ad = 10

; constants in enum type: t_cgep_element_type
   c_cgep_float_elm = 0
   c_cgep_double_elm = 1
   c_cgep_complexS_elm = 2
   c_cgep_complexD_elm = 3
   c_cgep_int16_elm = 4
   c_cgep_int32_elm = 5

;  Data Structure for Control block
TMP={t_cgep_control_block, $
           cpi_number:0L, $
           number_of_samples:0L, $
           prf:0.0, $
           offset:0L }

;  Data Structure for Header block
;
TMP={t_cgep_header_block, $
           filetype:0L, $
           satdef_pathname:BYTARR(256), $
           envdef_pathname:BYTARR(256), $
           elm_type:0L, $
           elm_size:0L, $
           control_block_offset:0L, $
           data_block_offset:0L, $
           number_of cpis:0L, $
           control_block: REPLICATE({t_cgep_control_block}, c_sgen_max_cpis)}

;
;  Product handle data structure
;  This structure contains the unit number of Intermediate Product,
;  header block info, and control block info.
;

TMP={t_rssir_prod_handle, $
                file_name:"", $
                unit:0L, $
                header: {t_cgep_header_block}, $
                cntr_blk: {t_csip_DI_cntr_blk}}

; release data memory
TMP=0L
```

```
; ==============================================================================
;
; SPD - Signal Processing Demonstrator Computer System
;
; (C) Copyright Her Majesty the Queen in Right of Canada
; as represented by the Minister of National Defence
;
; File Name:    cdat_rssir_rd.pro - Read RSS Intermediate Products
;                                   in PV-Wave
;
;
; Identification:        [@(#)]
;
; SCCS Id:      %W% %E%
;
; Responsible Organization:
;
;               Raytheon Canada Limited
;
; Purpose:      This file implements ingest functions in PV-Wave command
;               language to read RSS Intermediate Product data
;               files in XDR format.
;
;               These files are written out using the file format described
;               in the CGEP package.  The Intermediate Product files
;               will contain a header block, a control block and
;               the data block.
;
; Designer:     Kerry Zielke
;
; Change History:
;
;    SCR   Date        Author    Change made
;    --- ----------- ------  ------------------------------------------
;    ??? 22-Oct-1993  kmz    Initial Release
;
```

```
; ============================================================================
;
; ============================================================================
;    R E A D   U T I L I T Y   F U N C T I O N S
;
; ============================================================================
; FUNCTION:    RSSIR_RD_STRING
;
;   Description:
;       This function reads a string from an XDR file. The string size in
;       characters is passed to this function.  The string size can
;       be known by applying the "SIZE" function to the string.
;       The returned string is defined as a byte array with the specified
;       string size.
;
;   Synopsis:
;       string = RSSIR_RD_STRING (prod_handle, str_len)
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         RSSIR_OPEN
;
;       str_len                 - in    - string size in characters
;       string                  -   out - a string defined in byte array
;
FUNCTION RSSIR_RD_STRING, prod_handle, str_len
   str=BYTARR(str_len)
   char=0L
   FOR i=0, str_len-1 DO BEGIN
      READU, prod_handle.unit, char
      str(i)=BYTE(char,3)
   END
   RETURN, str
END     ; end of RSSIR_RD_STRING
```

```
; =========================================================================
; FUNCTION:     RSSIR_RD_FLOAT
;
;   Description:
;       This function reads a single-precision floating point value
;       from an XDR file.
;
;   Synopsis:
;       float_val = RSSIR_RD_FLOAT (prod_handle)
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         RSSIR_OPEN
;       float_val               -   out - floating pointing value
;
FUNCTION RSSIR_RD_FLOAT, prod_handle
   float_val=0.0
   READU, prod_handle.unit, float_val
   RETURN, float_val
END     ; end of RSSIR_RD_FLOAT
```

```
; ============================================================================
; FUNCTION:    RSSIR_RD_LONG
;
;   Description:
;       This function reads a longword integer (32 bytes)
;       from an XDR file.
;
;   Synopsis:
;       lonint_val = RSSIR_RD_LONG (prod_handle)
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         RSSIR_OPEN
;       lonint_val              -   out - longword integer value
;
FUNCTION RSSIR_RD_LONG, prod_handle
   long_val=0L
   READU, prod_handle.unit, long_val
   RETURN, long_val
END     ; end of RSSIR_RD_LONG
```

```
; ==========================================================================
;
;    OPEN/CLOSE   FUNCTIONS
;
; ==========================================================================
; PROCEDURE:    RSSIR_OPEN
;
;   Description:
;       Open a RSS Intermediate Product file in XDR format.
;       The logical file unit is allocated by this procedure and will be
;       stored in the returned product handle.
;
;       This procedure reads the header block and the control block,
;       and then stores them into a product handle structure,
;       which is defined in cdat_rssir_intf.pro.  This product handle is
;       then passed to the subsquent calls for reading individual Data
;       Update Intervals.
;
;   Synopsis:
;       RSSIR_OPEN, file_name, prod_handle
;
;   Parameters:
;       file_name               - in    - pathname of Intermediate product
;                                         file
;
;       prod_handle             -   out - product handle structure which
;                                         contains both the header block and
;                                         control block information.
;
PRO RSSIR_OPEN, file_name, prod_handle
   hdl={t_rssir_prod_handle}


;
; store the file name into the product handle
;
   hdl.file_name=file_name
   hdl.unit=-1  ; indicates the file has not been opened
   ON_IOERROR, bail_out
;
; open file with XDR keyword
; use the GET_LUN keyword to allocate a logical file unit
;
   OPENR, unit, file_name, /XDR, /GET_LUN

; store the logical file unit into the product handle
   hdl.unit=unit
   GOTO,  ok
;
; in case of opening error, it will print the error message.
;
bail_out:
   PRINT, !ERR_STRING
   prod_handle=hdl
   RETURN


ok:
   prod_handle=hdl
;   PRINT, 'file name: ', prod_handle.file_name
;   PRINT, 'logical file unit: ', prod_handle.unit
```

```
   header_blk={t_cgep_header_block}
;
;   read header block into memory and store it into product handle
;
  header_blk.filetype = DSPIR_RD_LONG(prod_handle)
  sz = SIZE(header_blk.satdef_pathname)
  header_blk.satdef_pathname=DSPIR_RD_STRING(prod_handle, sz(1))
  sz = SIZE(header_blk.envdef_pathname)
  header_blk.envdef_pathname=DSPIR_RD_STRING(prod_handle, sz(1))
  header_blk.elm_type = DSPIR_RD_LONG(prod_handle)
  header_blk.elm_size = DSPIR_RD_LONG(prod_handle)
  header_blk.control_block_offset = DSPIR_RD_LONG(prod_handle)
  header_blk.data_block_offset = DSPIR_RD_LONG(prod_handle)
  header_blk.number_of_cpis = DSPIR_RD_LONG(prod_handle)

;    PRINT, "header_block:"
;    PRINT, "  filetype:", header_blk.filetype
;    PRINT, "  satdef_pathname: ", string(header_blk.satdef_pathname(*))
;    PRINT, "  envdef_pathname: ", string(header_blk.envdef_pathname(*))
;    PRINT, "  elm_type:", header_blk.elm_type
;    PRINT, "  elm_size:", header_blk.elm_size
;    PRINT, "  control_block_offset:", header_blk.control_block_offset
;    PRINT, "  data_block_offset:", header_blk.data_block_offset


;
;   read control block into memory and store it into product handle
;
;    PRINT, "control_block:"

    prod_handle.header = header_blk

; Read in number_of_cpis control blocks

  cntr_block_pntr = ASSOC(unit, $
                          {t_cgep_control_block},  $
                          header_blk.control_block_offset)

  FOR i=0, prod_handle.number_of_cpis DO BEGIN
     prod_handle.control_block(i) = cntr_block_pntr(i)
  ENDFOR

  RETURN
END     ; end of RSSIR_OPEN
```

```
; ===========================================================================
; PROCEDURE:    RSSIR_CLOSE
;
;   Description:
;       close a RSS Intermediate Product file.
;
;   Synopsis:
;       RSSIR_CLOSE, prod_handle
;
;   Parameters:
;       prod_handle             - in    - product handle structure returned
;                                         by RSSIR_OPEN.
PRO RSSIR_CLOSE, prod_handle
   IF prod_handle.unit GT 0 THEN BEGIN
      ; close and free the I/O unit
      FREE_LUN, prod_handle.unit
   END
   prod_handle.unit=-1  ; indicates the file has been closed
END     ; end of RSSIR_CLOSE
```

```
; ==========================================================================
;
;    R E A D   F U N C T I O N S
;
; ==========================================================================


; ==========================================================================
; FUNCTION:    RSSIR_RD_CPI
;
;   Description:
;       Given a CPI Identifier,this function reads the selected CPI data
;       from the product file into memory.
;
;       Remember that the file offset to the individual Data Update Interval
;       is stored in the control block.  Once the header block and control
;       block of the product file is read, any Data Update Interval
;       can then be accessed randomly from the file.
;
;   Synopsis:
;       cpi_data = RSSIR_RD_CPI (prod_handle, CPI_id)
;
;   Exceptions:
;
;   Note:
;       This function does NOT return the size and element data type of
;       "cpi_data" to the caller; however, these information can be obtained
;       by calling either RSSIR_SIZE function provided in this package or
;       PV-Wave SIZE function.
;
;   Parameters:
;       prod_handle            - in    - product handle returned from
;                                        RSSIR_OPEN
;       CPI_id                 - in    - CPI Identifier
;       cpi_data               -   out - a CPI Data Update Interval.
;                                        It is declared as 1D, 2D or 3D array
;                                        of either floating-point, complex or
;                                        structure data type.
;
FUNCTION RSSIR_RD_CPI, prod_handle, CPI_id

   cpi_data = -1
   unit = prod_handle.unit

   IF (CPI_id LE prod_handle.header.number_of_cpis) $
     THEN BEGIN
       offset = prod_handle.control_block(CPI_id).offset
       number_of_samples = prod_handle.control_block(CPI_id).number_of_samples
;
;    associate the element array using the offset stored in the control block.
;    the dimension of the array is defined by dimX, dimY and dimZ stored in
;    the control block.
;
       CASE prod_handle.header.elm_type OF
       ; c_cgep_float_elm
       0: BEGIN
          record = ASSOC (unit, FLTARR(num_of_samples), offset)
          cpi_data = record(0)
          END
```

```
      ; c_cgep_double_elm
      1: BEGIN
         record = ASSOC (unit, DBLARR(number_of_samples), offset)
         cpi_data = record(0)
         END

      ; c_cgep_complexS_elm
      2: BEGIN
         record = ASSOC (unit, COMPLEXARR(number_of_samples), offset)
         cpi_data = record(0)
         END

      ; c_cgep_complexD_elm
      3: BEGIN
         record = ASSOC (unit, COMPLEXARR(number_of_samples), offset)
         cpi_data = record(0)
         END

      ; c_cgep_int16_elm
      4: BEGIN
         record = ASSOC (unit, INTARR(number_of_samples), offset)
         cpi_data = record(0)
         END

      ; c_cgep_int32_elm
      5: BEGIN
         record = ASSOC (unit, LONARR(number_of_samples), offset)
         cpi_data = record(0)
         END

      ELSE: BEGIN
            PRINT, "invalid element type."
            END
      ENDCASE
      GOTO, return
    ENDIF

    PRINT, "the specified CPI_id: ", CPI_id, $
           " is not captured in the product file (", $
           prod_handle.file_name, ")"

return:
  RETURN, cpi_data
END     ; end of RSSIR_RD_CPI
```

```
; ==========================================================================
;
;    E N Q U I R Y    F U N C T I O N S
;
; ==========================================================================
;
; FUNCTION:    RSSIR_FILE_TYPE
;
;   Description:
;       Return the particular file type of RSS Intermediate Product given a product
;       handle.
;
;   Synopsis:
;       IR_type = RSSIR_PROD_TYPE (prod_handle)
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         RSSIR_OPEN
;
;       file_type                       -   out - file type (enum data
;                                         type)
;                                       c_cgep_rss_clut = 0
;                                       c_cgep_rss_exn = 1
;                                       c_cgep_rss_jam = 2
;                                       c_cgep_rss_tar = 3
;                                       c_cgep_rss_comp = 4
;                                       c_cgep_rss_post_nl = 5
;                                       c_cgep_rss_if_filt = 6
;                                       c_cgep_rss_post_sd = 7
;                                       c_cgep_rss_ant_patrn = 8
;                                       c_cgep_rss_filt_rsp = 9
;                                       c_cgep_rss_post_ad = 10
;
FUNCTION RSPIR_FILE_TYPE, prod_handle
   RETURN, prod_handle.header.filetype
END     ; end of RSPIR_FILE_TYPE
```

```
; ========================================================================
; PROCEDURE:     RSSIR_CPI_IDARR
;
;   Description:
;       For the CPI interval products, given a product handle, a Dwell
;       identifier and a NCI identifier, this function returns the
;       information of CPI Data Update Intervals captured.
;       The returned information includes the total number of CPI
;       Intervals captured and their identifiers.
;
;   Synopsis:
;       RSSIR_CPI_IDARR, prod_handle, cpi_num, cpi_idarr, cpi_countarr
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         DSPIR_OPEN
;       cpi_num                 -   out - the total number of CPI Intervals
;       cpi_idarr               -   out - an array containing the CPI Interval
;                                         Identifiers (integer number starting
;                                         from 1).
;                                         The size of this array is cpi_num.
;       cpi_countarr            -   out - an array containing the number of samples
;                                         in each CPI Interval

PRO RSSIR_CPI_IDARR, prod_handle, cpi_num, cpi_idarr, cpi_countarr
  cpi_num = 0
  cpi_idarr = -1
  cpi_countarr = -1
;
; declare size of cpi_idarr and cpi_countarr (array of longword integer)
;
  cpi_num = prod_handle.header.number_of_cpis
  cpi_idarr = INTARR(cpi_num)
  cpi_countarr = INTARR(cpi_num)

  FOR i=0, cpi_num-1 DO BEGIN
    cpi_idarr(i) = prod_handle.control_block(i).cpi_number
    cpi_countarr = prod_handle.control_block(i).number_of_samples
  ENDFOR

return:
  RETURN
END     ; RSSIR_CPI_IDARR
```

```
; ==========================================================================
; PROCEDURE:    RSSIR_DATA_TYPE

    Description:
;       This procedure returns: data element type and data size obtained from
;       the product handle.
;
;   Synopsis:
;       RSSIR_DATA_TYPE, prod_handle,  data_type, data_size
;
;   Parameters:
;       prod_handle             -   in  - product handle returned from RSSIR_OPEN
;       data_type               -   out - data element type code.
;                                         code = -1:    unknown type
;                                         code = 0:     floating-point
;                                                       (single-precission)
;                                         code = 1:     double-precision
;                                                       floating
;                                         code = 2:     single-precision complex
;                                         code = 3:     double-precision complex
;                                         code = 4:     16 bit integer
;                                         code = 5:     32 bit integer
;
;                                         NOTE:  PV_Wave does not support
;                                                double precision complex.
;       data_size               -   out - data size
;
 RO RSSIR_SIZE, prod_handle, data_type, data_size
  data_type = prod_handle.header.elm_type
  data_size = prod_handle.header.elm_size
RETURN
END     ; end of RSSIR_DATA_TYPE
```

```
; ============================================================================
; PROCEDURE:    RSSIR_PRINT
;
;   Description:
;       This procedure to print out the contents of the header block
;       and the control block.  This is for testing purpose.
;
;   Synopsis:
;       RSSIR_PRINT, prod_handle
;
;   Parameters:
;       prod_handle             - in    - product handle returned from
;                                         RSSIR_OPEN
;
;
PRO RSSIR_PRINT, prod_handle
   header_blk = prod_handle.header
   PRINT, "header_block:"
   PRINT, "  filetype:", header_blk.filetype
   PRINT, "  satdef_pathname: ", string(header_blk.satdef_pathname(*))
   PRINT, "  envdef_pathname: ", string(header_blk.envdef_pathname(*))
   PRINT, "  elm_type:", header_blk.elm_type
   PRINT, "  elm_size:", header_blk.elm_size
   PRINT, "  control_block_offset:", header_blk.control_block_offset
   PRINT, "  data_block_offset:", header_blk.data_block_offset

   PRINT, "control_block:"
   FOR i=0, header_blk.number_of_cpis-1 DO BEGIN
      PRINT, prod_handle.control_block(i).cpi_number
   ENDFOR
   RETURN
END     ; end of RSSIR_PRINT
```

D1.8 Satellite/Radar Definition (CRAD)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*===================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      crad_intf.x  - satellite/RAdar generation Definition

                CRAD is a package under the DMT [Data ManagemenT] CSC
                in the Controller Functional area. CRAD specifies the
                Satellite/Radar Signal Generation Definition contained
                in a data store (file) in the Controller Functional area.

Identification: [@(#)]

SCCS Id:        @(#)crad_intf.x 1.4 93/06/09

Responsible Organization:

                MacDonald Dettwiler & Associates

Purpose:        This package declares and encapsulates the satellite
                Radar Generation Simulation Run
                Definition in the XDR format.

                This file is written in RPC data description language in XDR
                (eXternal Data Representation) standard. This file will use
                "rpcgen" utility to generate the following two files:

                crad_intf.h     - global include file for programs reading
                                  or writing the Satellite/Radar Generation
                                  Definition data file to include;
                crad_intf_xdr.c - XDR source file which contains required XDR
                                  conversion routines to support the I/O
                                  service functions in crad_io.c.
Designer:       Rob Gottschalk

Change History:

    SCR  Date        Author    Change made
    ---  ----------- ------  ----------------------------------------
    n/a  11-Jan-1993  robg   Initial Release
         22-Feb-1993  cst    overall update
         02-May-1993  cst    added freq_mid and band_width in concept struct
                             removed far_SL
                             removed ant_pathname from t_crad_antenna_model
                             added new structures for reusable antenna model
                             file format:
                               t_crad_RRA_file and t_crad_CFA_file
                             removed sat_ident from t_crad_sat_def
                             made changes to reflect SAT HMI table
                             dated Mar.23, 1993
                               changed t_crad_IQ_imbalance to
                                       t_crad_IQ_imbalance
                               removed amplitude_none and
                                       added DC_offset_I and DC_offset_Q
                               removed capt_target and capt_jammer
                               removed RSS_spec_seed and noise_seed
                                       from t_crad_rxfe
                               removed comb_range_ambig and ant_pattern_extent
                                       from t_crad_gen_options
                               added earth_clutter_factor and
                                       jammer_noise_factor and
                                       clutter_pw_thrshd into
                                       t_crad_gen_optons
                               removed capt_post_fe, capt_post_sync from
                                       t_crad_data_capture
                               added capt_post_nl, capt_post_filt,
                                     capt_post_sd, capt_post_ad, capt_ant,
                                       capt_filt_rsp to t_crad_data_capture
                                 added RMS_aper_dist_err, pattern_RSS_gen,
                                       rx_pattern_pathname and
                                       tx_pattern_pathname to
                                       t_crad_CFA_antenna and t_crad_RRA_antenna
        08-May-1993    cst    moved concept field from t_crad_antenna_model
                                     to t_crad_sat_def to make it consistent
                                     with the HMI diague
                              fixed typo: mulipath to multipath
                              removed analog_effects and data_capture data
                                     elements for main antenna from
                                     t_crad_CFA_antenna because the analog
                                     effects and capture options are
                                     applicable for phase centres for CFA

=====================================================================*/

#ifdef RPC_HDR
%#ifndef crad_intf_h
%#define crad_intf_h
%#include "spd_intf.h"
#endif

/*---------------------------------------------------------------------------**
**                                                                           **
/*      DEFINE CONSTANTS                                                     **
**                                                                           **
**___________________________________________________________________________*/
#define c_crad_max_multipath_scat       10      /* max number of multipath
                                                   scatterers */


%/*-------------------------------------------------------------------------------*
% *                                                                              *
% *     DEFINE DATA STRUCTUES                                                    *
% *                                                                              *
% *------------------------------------------------------------------------------*/
%/*
% *   data structure for satellite angles
% */
struct t_crad_azim_graz
{
    t_spd_real              azimuth;        /* azimuth angle */
    t_spd_real              grazing;        /* grazing angle */
};

struct t_crad_azim_elev
{
    t_spd_real              azimuth;        /* azimuth angle */
    t_spd_real              elevation;      /* elevation angle */
};

struct t_crad_CFA_attitude                  /* SAT HMI 2.2.5 */
{
    t_spd_real              roll;           /* roll angle */
    t_spd_real              pitch;          /* pitch angle */
    t_spd_real              yaw;            /* yaw angle */
};

struct t_crad_RRA_attitude
{
    t_spd_real              roll;           /* roll angle */
                                            /* SAT HMI 2.3.15 */
    t_spd_real              pitch;          /* pitch angle */
                                            /* SAT HMI 2.3.15 */
    t_spd_real              elevation;      /* elevation angle */
                                            /* SAT HMI 2.3.16 */
};
%/*-------------------------------------------------------------------------
% *
% *   locations in X and Y direction
% */
struct t_crad_xy {
   t_spd_real           x;              /* value in x direction */
   t_spd_real           y;              /* value in y direction */
};

%/*------------------------------------------------------------------------
% *
% *   locations in X, Y and Z direction
% */
struct t_crad_xyz {
   t_spd_real           x;              /* value in x direction */
   t_spd_real           y;              /* value in y direction */
   t_spd_real           z;              /* value in z direction */
};


%/*------------------------------------------------------------------------
% *
% *   Radar Concept data structure - defines the radar concept definition
% */
struct t_crad_concept
{
   t_spd_concepts       concept;        /* radar concept */
                                        /* SAT HMI 1.1 */
   t_spd_bands          freq_band;      /* frequency band (L or S) */
                                        /* SAT HMI 1.1.1 */
   t_spd_real           freq_mid;       /* mid-band frequency */
                                        /* SAT HMI 1.1.2 */
   t_spd_real           bandwidth;      /* RF bandwidth */
                                        /* SAT HMI 1.1.2 */
                                        /* NOTE: freq_min and freq_max are
                                                 computed from freq_mid and
                                                 band_width */
   t_spd_real           freq_min;       /* min frequency range for band */
   t_spd_real           freq_max;       /* max frequency range for band */
};

%/*------------------------------------------------------------------------
% *
% *   Signal Generation Options - defines the signal generation options
% */
struct t_crad_gen_options                              /* SAT HMI 7 */
{
   t_spd_int32          clutter_factor;                /* earth clutter oversampling
                                                          factor */
                                                       /* SAT HMI 7.1 */
   t_spd_int32          jammer_noise_factor;           /* jammer and noise oversampling
                                                          factor */
                                                       /* SAT HMI 7.1 */
   t_spd_real           clutter_pw_thrshd;             /* earth clutter power
                                                          threshold */
                                                       /* SAT HMI 7.2 */
   t_spd_real           range_quant_factor;            /* range quantization factor */
                                                       /* SAT HMI 7.3 */
   t_spd_real           doppler_quant_factor;          /* doppler quantization
                                                          factor */
                                                       /* SAT HMI 7.4 */
};


%/*------------------------------------------------------------------------
% *
% *   orbital position data structure - defines the orbital position definition
% */
%/* referenced Sub-satellite point (SSP) location in beam centre
%    lat/long coords and its azimuth and grazing angles */

struct t_crad_SSP_ref_loc               /* SAT HMI 2.1.4 derived method */
{
   t_spd_location       beam_centre;    /* Beam centre lat/lon */
   t_crad_azim_graz     azimuth_grazing;/* Beam centre angles: azimuth/grazing
                                           angles */
};

struct t_crad_orbital_position
{
   t_spd_real           altitude;       /* satellite altitude */
                                        /* SAT HMI 2.1.1 */
   t_spd_real           incl_angle;     /* angle of inclination */
                                        /* SAT HMI 2.1.2 */
   t_spd_UTC            initial_TOD;    /* initial time of day */
                                        /* SAT HMI 2.1.3 */
   t_spd_sat_dir        direction;      /* satellite approach direction */
                                        /* SAT HMI 2.1.5 */

   /* Initial Sub-Satellite Point location (either direct location or derived
      from beam location) */
                                        /* SAT HMI 2.1.4 */
   t_spd_location_methods loc_method;   /* position location method */
   t_spd_location       SSP_loc;        /* direct satellite location in
                                           lat/long coords */
                                        /* direct method */
   t_crad_SSP_ref_loc   ref_loc;        /* referenced location in beam centre
                                           lat/long coords and its azimuth
                                           and grazing angles */
                                        /* derived method */
};

%/*---------------------------------------------------------------------------
% *
% *   Analog Effects data structure - defines the antenna model
% *   analog effects
% */
%/*
% * Non-Linear character
% */
struct t_crad_non_linear {              /* SAT HMI 2.5.5.2 */
   t_spd_non_linear_char_types  character_type;
                                        /* Non-linear character */
                                        /* SAT HMI 2.5.5.2 */
   t_spd_real           parm[4];        /* user-entered parameters */
};
%/*
% * user-entered mismatch for IF & Equalizer Character
% */
struct t_crad_mismatch {                /* SAT HMI 2.5.5.3.1 */
   t_spd_real           amplitude;      /* amplitude mismatch (dB) */
   t_spd_real           phase;          /* phase musmatch (degrees) */
   t_spd_int32          ripples;        /* mismatch ripples (integer) */
};
%/*
% * IF & Equalizer Character
% */
struct t_crad_IF_equalization {
   t_spd_IF_equal_types equalization_type;
                                        /* IF & Equalization character type */
                                        /* SAT HMI 2.5.5.3 */
   t_crad_mismatch      mismatch;       /* user-entered mismatch only if
                                           equalization_type =
                                           c_spd_user_entered_mismatch */
                                        /* SAT HMI 2.5.5.3.1 */
   char                 IF_pathname[c_spd_path_size];
                                        /* user-specified stored file only if
                                           equalization_type =
                                           c_spd_user_defined_IF_file */
};
%/*
% * A/D character
% */
```

```
struct t_crad_AD_character {                   /* SAT HMI 2.5.5.4 */
    t_spd_AD_char_types  AD_type;              /* A/D character type */
                                               /* SAT HMI 2.5.5.4 */
    t_spd_real           coef[10];             /* user-entered coefficients */
                                               /* only if AD_type =
                                                  c_spd_user_defined_AD */
                                               /* SAT HMI 2.5.5.4.1 */
    char                 AD_pathname[c_spd_path_size];
                                               /* user-provide model file */
                                               /* only if AD_type =
                                                  c_spd_provided_AD_model */
};
%/*
% * IQ Imbalance
% */
struct t_crad_IQ_imbalance {                   /* SAT HMI 2.5.5.5 */
    t_spd_real           amplitude;            /* amplitude */
    t_spd_real           phase;                /* phase */
    t_spd_real           DC_offset_I;          /* DC offset - I (bits) */
    t_spd_real           DC_offset_Q;          /* DC offset - Q (bits) */
};
%/*
% * analog effect data structure
% */
struct t_crad_analog_effects
{
    t_spd_real           noise_temp;           /* noise temperature */
    t_crad_non_linear    non_linear_char;/* non-linear character */
    t_crad_IF_equalization equalizer_char;/* equalizer character */
    t_crad_AD_character  AD_char;              /* A/D character */
    t_crad_IQ_imbalance  IQ_imbalance;         /* I/Q imbalance */
};

%/*-----------------------------------------------------------------------
% *
% *   Data Capture data structure - defines the antenna model
% *   data capture options
% */
struct t_crad_data_capture
{                                      /* all flags are on/off */
    t_spd_boolean          capt_clutter;    /* capt clutter */
                                            /* SAT HMI 6.1 */
    t_spd_boolean          capt_ext_noise;  /* capt external noise */
                                            /* SAT HMI 6.2 */
    t_spd_boolean          capt_jammer;     /* capt jammer */
                                            /* SAT HMI 6.3 */
    t_spd_boolean          capt_target;     /* capt target */
                                            /* SAT HMI 6.4 */
    t_spd_boolean          capt_composite;  /* capt composite signal */
                                            /* SAT HMI 6.5 */
    t_spd_boolean          capt_post_nl;    /* capt post-Non-Linearity signal */
                                            /* SAT HMI 6.6 */
    t_spd_boolean          capt_post_filt;  /* capt iF-Filter output signal */
                                            /* SAT HMI 6.7 */
    t_spd_boolean          capt_post_sd;    /* capt post-synchronous detector
                                               signal */
                                            /* SAT HMI 6.8 */
    t_spd_boolean          capt_post_ad;    /* capt A/D output signal */
                                            /* SAT HMI 6.9 */
    t_spd_boolean          capt_ant;        /* capt antenna pattern */
                                            /* SAT HMI 6.10 */
    t_spd_boolean          capt_filt_rsp;   /* capt IF-Filter response */
                                            /* SAT HMI 6.11 */

};

%/*-----------------------------------------------------------------------
% *
% *   Receiver Front End data structure - defines the antenna model
% *   receiver front end
% */
struct t_crad_rxfe
{
    t_spd_real            bandwidth;       /* front end receiver band width */
                                           /* (MHz) */
                                           /* SAT HMI 2.5.1 */
    t_spd_real            sampling_rate;   /* sampling rate */
                                           /* (MHz) */
                                           /* SAT HMI 2.5.2 */
    t_spd_polarization    polarization;    /* polarization (H/V) */
                                           /* SAT HMI 2.5.3 */
    t_spd_boolean         motion_comp;     /* motion compensation (on/off) */
                                           /* SAT HMI 2.5.4 */
    t_spd_real            main_beam_delay;/* main beam delay (nsec) applied to
                                              main antenna or all phase centres
                                              such that the main antenna/PCs
                                              are centred in time relative
                                              to the anxiliary and its delay
                                              taps */
                                           /* SAT HMI 2.5.6 */
    t_spd_pulse_types     pulse_type;      /* waveform pulse type */
                                           /* SAT HMI 2.5.7 */

};

%/*---------------------------------------------------------------------
% *
% *   Multipath scatter data structure - defines each multipath
% *   scatter
% */
struct t_crad_multipath_scat {
    t_crad_xyz            position;        /* position in X, Y, and Z */
                                           /* SAT HMI 2.6.1 */
    t_spd_real            avg_reflectivity;/* average reflectivity (dB) */
                                           /* SAT HMI 2.6.2 */
    t_spd_real            avg_lobe_width;  /* average lobe width (degrees) */
                                           /* SAT HMI 2.6.3 */

};

%/*---------------------------------------------------------------------
% *
% *   Common data types for defining CFA/RRA antenna data structure
% */
/* analog effects for all phase centres within a DPCA group */
typedef t_crad_analog_effects t_crad_ph_ctr_analog_effects[c_spd_max_ph_ctrs];

/* data capture options for all phase centres within a DPCA group */
typedef t_crad_data_capture   t_crad_ph_ctr_data_capture[c_spd_max_ph_ctrs];

/* element pattern pathname (directory + file name) */
typedef char t_crad_pattern_pathname[c_spd_path_size];
typedef t_crad_pattern_pathname t_crad_ph_ctr_patn_name[c_spd_max_ph_ctrs];


%/*---------------------------------------------------------------------
% *
% *   CFA antenna data structure - define a main antenna in CFA concept
% */
struct t_crad_CFA_antenna /* for CFA Type Only */
{
    /* physical characteristics */
    t_crad_xy             aperture;        /* aperture - along motion (x)
                                              and across motion (y) */
                                           /* SAT HMI 2.2.1 */
    t_spd_element_grids   grid_type;       /* element grid type */
                                           /* SAT HMI 2.2.2 */
    t_crad_xy             grid_spacing;    /* grid spacing - dx and dy */
                                           /* SAT HMI 2.2.3 */
    t_spd_element_patterns pattern_type;   /* element pattern type */
                                           /* SAT HMI 2.2.4 */
    t_spd_real            horn_gain;       /* element horn gain */
```

```
                                                  /* SAT HMI 2.2.4.1 */

         t_spd_int32             num_DPCA_grps;  /* number of DPCA groups */
                                                  /* SAT HMI 2.2.6 */
         t_spd_int32             num_ph_ctr_per_grp;
                                                  /* number of phase Centres per
                                                     DPCA group */
                                                  /* SAT HMI 2.2.6.1 */
         t_spd_int32             PRI_multiplier; /* PRI multiplier */
                                                  /* SAT HMI 2.2.6.2 */
                                                  /* valid only if
                                                     num_ph_ctr_per_grp >= 2 */
         t_crad_xy               disp[c_spd_max_DPCA_grps];
                                                  /* displacement from aperture
                                                     centre - X & Y for each group */
                                                  /* SAT HMI 2.2.6.3 */

         t_spd_weighting_types rx_weight;        /* receive array weight */
                                                  /* SAT HMI 2.2.7 */
         t_spd_real              rx_near_SL;     /* receive near sidelob */
                                                  /* SAT HMI 2.2.7.1 */
         t_spd_weighting_types tx_weight;        /* transmit array weight */
                                                  /* SAT HMI 2.2.8 */
         t_spd_real              tx_near_SL;     /* transmit near sidelobe */
                                                  /* SAT HMI 2.2.8.1 */
         t_spd_real              RMS_aper_dist_err;
                                                  /* RMS aperture distrubution error */
                                                  /* SAT HMI 2.2.9 */
         t_spd_boolean           pattern_RSS_gen;/* RSS- or user-generated */
                                                  /* SAT HMI 2.2.10 */
         t_crad_pattern_pathname tx_pattern_pathname[c_spd_max_DPCA_grps];
                                                  /* Transmit pattern pathname */
                                                  /* total number of Tx antennae is
                                                     DPCA_grps */
                                                  /* SAT HMI 2.2.10.1 */
         t_crad_ph_ctr_patn_name rx_pattern_pathname[c_spd_max_DPCA_grps];
                                                  /* Receive pattern pathname */
                                                  /* total number of Rx antenne is the
                                                     total number of phase centres:
                                                     num_ph_ctr_per_grp*DPCA_grps */
                                                  /* SAT HMI 2.2.10.1 */

         /* operational characteristics */
         t_crad_CFA_attitude  attitude;          /* attitude: roll, pitch, yaw */
                                                  /* SAT HMI 2.2.5 */


         /* others */
         t_crad_ph_ctr_analog_effects ph_ctr_anal_efft[c_spd_max_DPCA_grps];
                                                  /* phase centre analog effects */
                                                  /* SAT HMI 2.5.5 */
         t_crad_ph_ctr_data_capture   ph_ctr_dat_capt[c_spd_max_DPCA_grps];
                                                  /* phase centre data capture */
                                                  /* SAT HMI 6 */

      };
      %/*------------------------------------------------------------------------
      % *
      % *   CFA antenna data structure - define a main antenna in RRA concept
      % */
      struct t_crad_RRA_antenna
      {
         /* physical characteristics */
               /* reflectors */
         t_spd_real              main_diameter;  /* main reflector diameter */
                                                  /* SAT HMI 2.3.1 */
         t_spd_real              main_focal_len; /* main reflector focal length */
                                                  /* SAT HMI 2.3.2 */
         t_spd_real              sub_diameter;   /* sub reflector diameter */
                                                  /* SAT HMI 2.3.3 */
```

```
    t_spd_real           sub_focal_len;  /* sub reflector focal length */
                                         /* SAT HMI 2.3.4 */

    t_spd_real           feed_diameter;  /* antenna feed diameter */
                                         /* SAT HMI 2.3.5 */
    t_spd_real           feed_axial_pos; /* antenna feed axial position */
                                         /* SAT HMI 2.3.6 */

    t_spd_element_grids  feed_grid_type; /* feed element grid type */
                                         /* SAT HMI 2.3.7 */
    t_crad_xy            feed_grid_spacing;
                                         /* feed grid spacing - dx and dy */
                                         /* SAT HMI 2.3.8 */
    t_spd_element_patterns feed_pattern_type;
                                         /* feed element pattern type */
                                         /* SAT HMI 2.3.9 */
    t_spd_real           horn_gain;      /* horn gain */
                                         /* SAT HMI 2.3.9.1 */

    t_spd_weighting_types tx_feed_weight;/* transmit array weight */
                                         /* SAT HMI 2.3.10 */
    t_spd_real           tx_feed_near_SL;/* transmit near sidelobe */
                                         /* SAT HMI 2.3.10.1 */
    t_spd_weighting_types rx_feed_weight;/* receive array weight */
                                         /* SAT HMI 2.3.11 */
    t_spd_real           rx_feed_near_SL;/* receive near sidelob */
                                         /* SAT HMI 2.3.11.1 */
    t_spd_real           RMS_aper_dist_err;
                                         /* RMS aperture distrubution error */
                                         /* SAT HMI 2.3.12 */
    t_spd_boolean        pattern_RSS_gen;/* RSS- or user-generated */
                                         /* SAT HMI 2.3.17 */
    t_crad_pattern_pathname tx_pattern_pathname;
                                         /* Transmit pattern pathname */
                                         /* SAT HMI 2.3.17.1 */
    t_crad_pattern_pathname rx_pattern_pathname;
                                         /* Receive pattern pathname */
                                         /* SAT HMI 2.3.17.1 */

    /* operational characteristics */
    t_spd_real           rotation_rate;  /* rotation rate (rpm) */
                                         /* SAT HMI 2.3.13 */
    t_spd_real           boresight_azimuth;
                                         /* boresight azimuth (degree) */
                                         /* SAT HMI 2.3.14 */
    t_crad_RRA_attitude  attitude;       /* attitude: roll, pitch, and
                                            elevation angle */
                                         /* SAT HMI 2.3.15 & 16 */

    /* others */
    t_crad_analog_effects analog_effects;/* main antenna analog effects */
                                         /* SAT HMI 2.5.5 */
    t_crad_data_capture  data_capture;   /* main antenna data capture */
                                         /* SAT HMI 6 */
};


%/*---------------------------------------------------------------------
% *
% *   Main Antenna data structure
% *   -  completely defines the antenna model definition
% */
struct t_crad_main_antenna
{
    t_crad_RRA_antenna RRA;              /* RRA antenna characteristics */
                                         /* SAT HMI 2.3 */
    t_crad_CFA_antenna CFA;              /* CFA antenna characteristics */
                                         /* SAT HMI 2.2 */
};

```

```

  %/*------------------------------------------------------------------------
  % *
  % *   Delay Tap data structure - defines a delay tap
  % *   for the auxiliary antenna definition
  % */
  struct t_crad_delay_tap
  {
     t_spd_real            tap_delay;      /* tap delay (nsec) */
                                           /* SAT HMI 2.4.5.1 */
     t_crad_data_capture  data_capture;    /* delay tap data capture */
                                           /* SAT HMI 6 */
  };

  %/*------------------------------------------------------------------------
  % *
  % *   Auxiliary antenna Regular Array data structure
  % *   - defines the auxiliary antenna regular array definition
  % */
  struct t_crad_aux_regular_arr
  {
     t_spd_boolean         tracking_flag;  /* tracking array flag */
                                           /* SAT HMI 2.4.6.1 */
     t_crad_azim_elev      tracking_offset;/* tracking array offset
                                              - arimuth and elevation angle */
                                           /* SAT HMI 2.4.6.2 */
     t_crad_xy             aperture;       /* aperture along motion - x & y */
                                           /* SAT HMI 2.4.6.3 */
     t_spd_element_grids  grid_type;       /* element grid type */
                                           /* SAT HMI 2.4.6.1 */
     t_crad_xy             grid_spacing;   /* grid spacing - dx & dy */
                                           /* SAT HMI 2.4.6.5 */
     t_spd_weighting_types array_weight;   /* array weight */
                                           /* SAT HMI 2.4.6.6 */
     t_spd_real            array_near_SL;  /* array near sidelob */
                                           /* SAT HMI 2.4.6.6.1 */
  };

  %/*------------------------------------------------------------------------
  % *
  % *   Auxiliary Antenna Arbitrary Array data structure
  % *   - defines the auxiliary antenna arbitrary array definition
  % */
  struct t_crad_aux_arbitrary_elm
  {
     /* for each element */
     t_crad_xyz            location;       /* location in x, y, z */
                                           /* SAT HMI 2.4.7.2.1 */
     t_spd_real            fixed_ph_shift; /* fixed phase shift */
                                           /* SAT HMI 2.4.7.2.2 */
     t_spd_real            fixed_amp_weight;/* fixed amplitude weight */
                                           /* SAT HMI 2.4.7.2.3 */
  };

  struct t_crad_aux_arbitrary_arr
  {
     t_spd_boolean         tracking_flag;  /* tracking array flag */
                                           /* SAT HMI 2.4.7.1 */
     t_spd_int32           elm_count;      /* number of elements */
                                           /* SAT HMI 2.4.7.2 */
     t_crad_aux_arbitrary_elm elm[c_spd_max_elm_arbtr_arr];
                                           /* SAT HMI 2.4.7.2.1 - 2.4.7.2.3 */
  };
  %/*------------------------------------------------------------------------
  % *
  % *   Auxiliary antenna data structure (Physical and operational)
  % *   -  defines the physical and operational characteristics of
  % *      the auxiliary antenna
  % */
```

```
struct t_crad_auxiliary_antenna
{
   /* physical characteristics */

   t_spd_array_types    aux_type;        /* single, regular or arbitrary */
                                         /* SAT HMI 2.4.1 */
   t_spd_element_patterns elm_pattern;   /* element pattern */
                                         /* SAT HMI 2.4.2 */
   t_spd_real           horn_gain;       /* horn gain */
                                         /* SAT HMI 2.4.2.1 */

   /* phase centre location */
   t_crad_xyz           ph_ctr_loc;      /* Auxiliary phase centre
                                            location in x, y, and z */
                                         /* SAT HMI 2.4.3 */

   t_spd_int32          tap_count;       /* number of delay taps within
                                            this auxilary antenna */
                                         /* SAT HMI 2.4.5 */
   t_crad_delay_tap     delay_taps[c_spd_max_taps];
                                         /* delay taps data structure */
                                         /* SAT HMI 2.4.5.1 */
   t_crad_aux_regular_arr   reg_arr;     /* regular array definition */
                                         /* valid only if
                                            aux_type = c_spd_regular_array */
                                         /* SAT HMI 2.4.6 */
   t_crad_aux_arbitrary_arr arb_arr;     /* arbitrary array definition */
                                         /* valid only if
                                            aux_type = c_spd_arbitary_array */
                                         /* SAT HMI 2.4.7 */

   /* operational characteristics */

   t_spd_site_types     site_type;       /* non-rotating or rotating */
   t_crad_CFA_attitude  fix_attitude;    /* Attitude for non-rotating auxiliary
                                            (RRA or CFA) */
                                         /* attitude: roll, pitch, yaw */
                                         /* SAT HMI 2.4.4 */
   t_crad_azim_elev     rot_attitude;    /* Attitude for rotating auxiliary
                                            (RRA only) */
                                         /* attitude: azimuth offset and
                                            elevation */
                                         /* SAT HMI 2.4.4 */

   /* others */
   t_crad_analog_effects analog_effects;/* auxiliary analog effects */
                                         /* SAT HMI 2.5.5 */
   t_crad_data_capture  data_capture;    /* auxiliary data capture option */
                                         /* SAT HMI 6 */
};



%/*----------------------------------------------------------------------------
% *
% *   Antenna Model data structure - completely defines the antenna model
% *   definition
% */
struct t_crad_antenna_model
{
   t_crad_main_antenna  main_antenna;    /* main antenna definition */
   t_spd_int32          aux_count;       /* number of defined auxiliaries */
   t_crad_auxiliary_antenna aux_antenna[c_spd_max_aux];
                                         /* auxiliary antennae definitions */
   t_crad_rxfe          main_rxfe;       /* main antenna receiver front end */
                                         /* SAT HMI 2.5 */
   t_crad_multipath_scat multipath_scat[c_crad_max_multipath_scat];
                                         /* multipath scatters */
                                         /* SAT HMI 2.6 */
};
```

```
%/*--------------------------------------------------------------------------
% *
% *   Antenna Model file structure - completely defines the reusable
% *   antenna model stored in an external file
% *
% *   RRA:      output t_crad_RRA_file structure
% *   CFA:      output t_crad_CFA_file structure
% */
struct t_crad_RRA_file          /* RRA reusable antenna model */
{
   t_crad_RRA_antenna   main_antenna;   /* main antenna definition */
                                        /* SAT HMI 2.3 */
   t_spd_int32          aux_count;      /* number of defined auxiliaries */
                                        /* SAT HMI 2.4 */
   t_crad_auxiliary_antenna aux_antenna[c_spd_max_aux];
                                        /* auxiliary antennae definitions */
   t_crad_rxfe          main_rxfe;      /* main antenna receiver front end */
                                        /* SAT HMI 2.5 */
   t_crad_multipath_scat multipath_scat[c_crad_max_multipath_scat];
                                        /* multipath scatters */
                                        /* SAT HMI 2.6 */
};
struct t_crad_CFA_file          /* CFA reusable antenna model */
{
   t_crad_CFA_antenna   main_antenna;   /* main antenna definition */
   t_spd_int32          aux_count;      /* number of defined auxiliaries */
   t_crad_auxiliary_antenna aux_antenna[c_spd_max_aux];
                                        /* auxiliary antennae definitions */
   t_crad_rxfe          main_rxfe;      /* main antenna receiver front end */
                                        /* SAT HMI 2.5 */
   t_crad_multipath_scat multipath_scat[c_crad_max_multipath_scat];
                                        /* multipath scatters */
                                        /* SAT HMI 2.6 */
};

%/*==========================================================================
% *
% *   CPI INTERVAL DEFINITION
% */
%/*--------------------------------------------------------------------------
% *
% *   CPI data structure - totally defines the CPI definition
% */
struct t_crad_CPI_elm
{
   t_spd_real           RF_freq;        /* RF frequency */
   t_spd_real           post_CPI_delay; /* post CPI delay */
   t_spd_boolean        capture_CPI_enable;
                                        /* yes/no flag indicating whether
                                           to capture the current CPI */
};


%/*==========================================================================
% *
% *   NCI INTERVAL DEFINITION
% */
%/*--------------------------------------------------------------------------
% *
% *   Data Reduction data structure - defines reduction options for all captured signal
% */
struct t_crad_data_reduction
{
   t_spd_decimation     time_decim;     /* time OR doppler decimation */
   t_spd_decimation     range_decim;    /* range decimation */
   t_spd_boolean        magnitude_sqrd; /* magnitude sqaured (on/off) */

};
```

```
%/*---------------------------------------------------------------------------
% *
% *   NCI data structure - defines a complete NCI definition (includes CPIs)
% */
struct t_crad_NCI_elm
{
   t_crad_CPI_elm       CPI<c_spd_max_CPI>;
                                         /* a list of CPI elements in the
                                            current NCI. It is defined as
                                            an array pointer and this array
                                            contains upto c_spd_max_CPI entries.
                                         */
                                         /* SAT HMI 4.2 */
   t_spd_boolean        PRF_entered;     /* indicates whether the user enters
                                            either PRF directly and in samples
                                            per PRI */
                                         /* SAT HMI 4.1.1 */
   t_spd_real           PRF;             /* pulse repetition frequency (Hz) */
   t_spd_int32          samples_per_PRI;/* number of samples per PRI */
   t_spd_int32          pulse_per_CPI;   /* pulses per CPI */
                                         /* SAT HMI 4.1.2 */
   t_spd_boolean        capture_NCI_enable;
                                         /* yes/no flag indicating whether
                                            to capture the current NCI */
                                         /* SAT HMI 3.2.1 */
};


%/*===========================================================================
% *
% *   DWELL INTERVAL DEFINITION
% */
%/*---------------------------------------------------------------------------
% *
% *   DWELL Interval waveform data structure - defines DI waveform
% */
struct t_crad_DI_waveform                /* SAT HMI 3.2 */
{
   t_spd_real           exp_pulse_dur;   /* expanded pulse duration */
                                         /* SAT HMI 3.2.1 */
   t_spd_real           compr_pulse_width; /* compressed pulse width */
                                         /* SAT HMI 3.2.2 */
   t_spd_boolean        QPM;             /* QPM on/off flag */
                                         /* SAT HMI 3.2.3 */
   t_spd_real           pulse_peak_power;/* peak power */
                                         /* SAT HMI 3.2.4 */

};

%/*---------------------------------------------------------------------------
% *
% *   DWELL Interval location data structure - defines DWI location
% */
struct t_crad_beam_location              /* SAT HMI 3.1 */
{
   t_spd_location_methods loc_method;    /* location method */
                                         /* SAT HMI 3.1.1 */
   t_spd_location       beam_centre;     /* specifying direct beam lat/long
                                            location */
   t_crad_azim_graz     azim_graz;       /* specifying azimuth & grazing
                                            angles */
   t_crad_azim_elev     boresight;       /* specifying boresight azimuth
                                            & elevation angles */
};


%/*---------------------------------------------------------------------------
% *
% *   Dwell element data structure - defines a complete dwell (includes NCIs)
% */
```

```
struct t_crad_DI_elm
{
   t_crad_NCI_elm        NCI<c_spd_max_NCI>;
                                             /* a list of NCI elements in the
                                                current DI. It is defined as
                                                an array pointer and this array
                                                contains upto c_spd_max_NCI entries.
                                             */
                                             /* SAT HMI 3.3 */
   t_crad_beam_location location;           /* dwell location parameters */
                                             /* SAT HMI 3.1 */
   t_crad_DI_waveform   waveform;           /* dwell waveform parameters */
                                             /* SAT HMI 3.2 */
   t_spd_boolean        capture_DI_enable;
                                             /* indicating whether to capture
                                                the current DI */
                                             /* SAT HMI 6.8.4 */

};


%/*=========================================================================
% *
% *   SATELLITE/RADAR SIGNAL GENERATION DEFINITION
% */
%/*------------------------------------------------------------------------
% *
% *   Satellite/Radar Signal Generation Definition - the complete signal generation definition
% */
struct t_crad_sat_def
{
   t_crad_concepts concept;                 /* concept structure */
                                             /* SAT HMI 1.1 */
   t_crad_orbital_position orbital_pos; /* orbital position definitions */
                                             /* SAT HMI 2.1 */
   t_crad_antenna_model antenna_model;  /* antenna model definition */
                                             /* SAT HMI 2.2 - 2.6 */
   t_crad_DI_elm        DI<c_spd_max_DI>; /* a list of DI elements for the
                                                current run. It is defined as
                                                an array pointer and this array
                                                contains upto c_spd_max_DI entries.
                                             */
   t_spd_boolean        capture_CTI_enable;
                                             /* yes/no flag indicating whether
                                                to capture the entire CTI (run) */
   t_crad_data_reduction data_reduction;/* data reduction options */
                                             /* SAT HMI 6.12 */
   t_crad_gen_options   gen_options;     /* signal generation options */
};

#ifdef RPC_HDR
%
%
%/*------------------------------------------------------------------------
% *
% *   SERVICE FUNCTIONS SPECIFICATION
% *
% *-----------------------------------------------------------------------*/
%/*=========================================================================
%
%   Function:    f_crad_read
%
%   Description:
%      This function reads a Satellite/Radar Definition data file
%      named by pathname into in-memory data structure.
%      pathname points to a character string that contains the directory and
%      the name of Satellite/Radar generation definition file to be read.
%      The memory for the data structure will be allocated by this function.
%      Therefore, it is the caller's responsibility to release the memory.
%
```

```
%   Arguments:
%      pathname                 - in    - directory and file name of
%                                          the Satellite/Radar
%                                          Definition data log.
%
%      sat_def                  -   out - Satellite/Radar Definition
%
%   External variables:
%      none
%
%   Exceptions:
%      s_spd_failure returned if the provided pathname is not found or
%      can not be read.
%
%   Notes:
%      none
%
%=============================================================================*/
%extern t_spd_status f_crad_read (
%                char *pathname,
%                t_crad_sat_def *sat_def);
%
%
%/*============================================================================
%
%   Function:   f_crad_write
%
%   Description:
%      This function writes a Satellite/Radar Definition data
%      structure to a data file named by pathname.
%      pathname points to a character string that contains the directory and
%      the name of Satellite/Radar generation definition file to be write.
%
%   Arguments:
%      pathname                 - in    - directory and file name of
%                                          the Satellite/Radar
%                                          Definition data log.
%      sat_def                  - in    - Satellite/Radar Definition
%
%   External variables:
%      none
%
%   Exceptions:
%      s_spd_failure returned if the data can not be written to the provided
%      pathname.
%
%   Notes:
%      none
%
%=============================================================================*/
%extern t_spd_status f_crad_write (
%                char *pathname,
%                t_crad_sat_def *sat_def);
#endif

#ifdef RPC_HDR
%#endif
#endif
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      crad_io.c - Satellite/Radar Definition I/O functions

Identification: [@(#)]

SCCS Id:        @(#)crad_io.c   1.2 93/05/11

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package implements read/write functions for the
                Satellite/Radar Definition data store.

Designer:       Stephen Tran

Change History:

   SCR  Date        Author   Change made
   ---  ----------  ------   ----------------------------------------
   n/a  29-Jan-1993  cst     Initial Release

==========================================================================*/

/* Included external declarations */
#include <stdio.h>
#include <rpc/rpc.h>
#include "spd_intf.h"
#include "cerr_intf.h"
#include "crad_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* Implementation */
/*==========================================================================

   Function:    f_crad_read

   Description:
      This function reads a Satellite/Radar Definition data file
      named by pathname into in-memory data structure.
      pathname points to a character string that contains the directory and
      the name of Satellite/Radar generation definition file to be read.
      The memory for the data structure will be allocated by this function.
      Therefore, it is the caller's responsibility to release the memory.

   Arguments:
      pathname                - in    - directory and file name of
                                          the Satellite/Radar
                                          Definition data log.
      sat_def                 -   out - Satellite/Radar Definition

   External variables:
      none

   Exceptions:
      s_spd_failure returned if the provided pathname is not found or
      can not be read.

   Notes:
      none

   High Level PDL
      BEGIN  f_crad_read
        USE fopen TO open Sat/Rad Definition file
        with "read" option
        IF file does not exist or can not be read THEN
           return s_spd_failure
        ELSE
           USE xdrstdio_create TO initialize a record-oriented XDR stream
           object (xdrs) in XDR_DECODE mode, which will be used in the
           subsquent XDR calls
           USE xdr_t_crad_sat_def TO read the Sat/Rad Definition
           file in XDR format into in-memory data structure
           IF error THEN return s_spd_failure
           USE xdr_destroy TO destroy xdrs
        ENDIF
      END f_crad_read
=========================================================================*/
t_spd_status f_crad_read (
                char *pathname,
                t_crad_sat_def *sat_def)
{
}


/*=========================================================================

   Function:    f_crad_write

   Description:
      This function writes a Satellite/Radar Definition data
      structure to a data file named by pathname.
      pathname points to a character string that contains the directory and
      the name of Satellite/Radar generation definition file to be write.

   Arguments:
      pathname                  - in    - directory and file name of
                                          the Satellite/Radar
                                          Definition data log.
      sat_def                   - in    - Satellite/Radar Definition

   External variables:
      none

   Exceptions:
      s_spd_failure returned if the data can not be written to the provided
      pathname.

   Notes:
      none

   High Level PDL
      BEGIN  f_crad_write
        USE fopen TO open Sat/Rad Definition file
        with "write" option
        IF file does not exist or can not be write THEN
           return s_spd_failure
        ELSE
           USE xdrstdio_create TO initialize a record-oriented XDR stream
           object (xdrs) in XDR_ENCODE, which will be used in the subsquent
           XDR calls
           USE xdr_t_crad_sat_def TO write the Sat/Rad Definition
           structure to data file in XDR format.
           IF error THEN return s_spd_failure
           USE xdr_destroy TO destroy xdrs
        ENDIF
      END f_crad_write
=========================================================================*/
t_spd_status f_crad_write (
                char *pathname,
                t_crad_sat_def *sat_def)
{
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

D1.9 Environment Definition (CEND)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*+ Identification cend_intf.x 1.1 93/09/09 15:44:41 */
%/*=========================================================================
%
%SPD - Signal Processing Demonstrator Computer System
%
   ) Copyright Her Majesty the Queen in Right of Canada
%as represented by the Minister of National Defence, 1992
%
%File Name:      cend.h  - Environment Definition export file
%
%                CEND is a package under the DMT [Data ManagemenT] CSC
%                in the Controller Functional area. CEND specifies the
%                Environment Definition that is extracted from a data
%                store in the Controller Functional area.
%
%Identification:         [@(#)]
%
%SCCS Id:        @(#)cend_intf.h 1.1 93/04/22
%
%Responsible Organization:
%
%               MacDonald Dettwiler and Associates
%
%Purpose:        The CEND package independently defines the environment
%                definition that can be associated with a signal generation
%                definition.
%
%Designer:       Stephen Tran
%
  hange History:
%
%    SCR   Date         Author    Change made
%    ---   -----------  ------  ------------------------------------------
%    ???   20-Jan-1993   cst    Initial Release
%
%========================================================================*/
%
#ifdef RPC_HDR
%#ifndef cend_intf_h
%#define cend_intf_h
%#if !defined lint
%static char cend_intf_h_sccsid[] = "@(#)cend_intf.x    1.1 93/09/09 15:44:41";/*% SCCS modul
%#include <rpc/types.h>
%#include "spd_intf.h"
%
%
%/*==========================================================================*/
%/*   C O N S T A N T   D E F I N I T I O N S                                */
%/*==========================================================================*/
%
%/*------------------------------------------------------------------------ **
%**                                                                         **
%**    Constants used in limiting the environment definitions               **
%**                                                                         **
%**------------------------------------------------------------------------*/

..define    c_cend_max_vertices  100     /* max clutter patch vertices */
%#define    c_cend_max_ftargets   10     /* max false targets per jammer */
%#define    c_cend_max_tbls      30      /* max table name length */
%#define    c_cend_max_patches    100    /* max clutter patches */
%#define    c_cend_max_discretes  100    /* max discretes */
%#define
```

```
%#define   c_cend_max_targets    100    /* max targets */
%#define   c_cend_max_jammers    100    /* max jammers */
%
%
#endif /* RPC_HDR */
%
%/*=====================================================================*/
%/*                                                                     */
%/*  D A T A   T Y P E S                                                */
%/*=====================================================================*/
%
%/*-------------------------------------------------------------------- **
%**                                                                     **
%**   Data structure for noise seed data                                **
%**                                                                     **
%**--------------------------------------------------------------------*/
%
struct t_cend_seed{ /* seed */
   t_spd_boolean        RSS_spec;       /* indicates whether the seed is
                                           RSS-specified or User-specified */

   t_spd_int32          seed;   /* user-enter integer range 1 to 2^32-1 */
};

%#define c_cend_min_seed_value 1
%#define c_cend_max_seed_value ULONG_MAX
%#define c_cend_default_seed_value 1
%
%/*-------------------------------------------------------------------- **
%**                                                                     **
%**   Data structure for External Noise model definition               **
%**                                                                     **
%**--------------------------------------------------------------------*/
%
struct t_cend_external_noise{ /* external noise model */
   t_spd_boolean        enabled;        /* external noise on or off */
   t_cend_seed      noise_seed;            /* external noise seed */

};
%
%/*-------------------------------------------------------------------- **
%**                                                                     **
%**   Data structure for Ionosphere model definition                   **
%**                                                                     **
%**--------------------------------------------------------------------*/
%
struct t_cend_ionosphere{ /* ionosphere model */
   t_spd_boolean        enabled;        /* ionosphere on or off */
   t_spd_real           electron_dens;  /* electronic density (e-3*m) */
   t_spd_real           alt_max_ion;    /* altitude of maximum ionization (km) */
   t_spd_real           scale_height;   /* scale heigh (km) */
   t_spd_real           scint_index;    /* scintillation index 0, 0.1, 0.2, 0.3, 0.4, 0.5
   t_cend_seed      ionsph_seed;            /* Scintillation seed */
};
%
%#define c_cend_default_enabled ON
%#define c_cend_min_electron_dens 1.0e9
%#define c_cend_max_electron_dens 1.0e14
%#define c_cend_min_alt_max_ion 100.0
%#define c_cend_max_alt_max_ion 300.0
%#define c_cend_min_scale_height 0.0
%#define c_cend_max_scale_height 50.0
%#define c_cend_default_scint_index 0.0
%                                                              ------- **
```

D-

```
%**                                                                          **
%**    Data Structure for Lower Atmosphere model defintiion                  **
%**                                                                          **
%**--------------------------------------------------------------------------*/

ruct t_cend_atmosphere{ /* lower atmosphere */
   t_spd_boolean        enabled;        /* atmosphere on or off */
   t_spd_intf_humid_types       humidity;       /* humidity type */
};
%
%/*------------------------------------------------------------------------- **
%**                                                                          **
%**    Data structure for wind data definition                               **
%**                                                                          **
%**--------------------------------------------------------------------------*/
struct t_cend_land{                /* land only */
  t_spd_real    spd;               /* wind speed (knots) */
};

%/*------------------------------------------------------------------------- **
%**                                                                          **
%**    Data structure for sea data definition                               **
%**                                                                          **
%**--------------------------------------------------------------------------*/
struct t_cend_sea{                                      /* sea only */
  t_spd_real    dir;               /* wind direction rsp to North (degrees) */
  t_spd_int32   sea_state;         /* sea state (Douglas) */
};

   *------------------------------------------------------------------------ **
   *                                                                         **
%**    Data structure for land data definition                               **
%**                                                                          **
%**--------------------------------------------------------------------------*/
union t_cend_wind
   switch ( t_spd_clutter_types clutter_type ) {
  case c_spd_sea :
     t_cend_sea       sea;

  default :
     t_cend_land        land;
};
%#define c_cend_min_spd 0.0
%#define c_cend_max_spd 100.0
%#define c_cend_default_spd c_cend_min_spd
%#define c_cend_min_direction 0.0
%#define c_cend_max_direction 360.0
%#define c_cend_default_direction c_cend_min_direction
%#define c_cend_sea_state_0 0
%#define c_cend_sea_state_1 1
%#define c_cend_sea_state_2 2
%#define c_cend_sea_state_3 3
%#define c_cend_sea_state_4 4
%#define c_cend_sea_state_5 5
%#define c_cend_default_sea_state c_cend_sea_state_3

%/*------------------------------------------------------------------------- **
%**                                                                          **
%**    Data structure for Background clutter model definition               **
%**                                                                          **
%**--------------------------------------------------------------------------*/
```

```
   t_spd_boolean        enabled;        /* background clutter on or off */
   t_spd_clutter_types  clutter_type;
   t_cend_wind          wind;           /* wind information */
};

%#define c_cend_default_clutter_type c_spd_baren
%
%/*------------------------------------------------------------------------- **
%**                                                                          **
%**    Data structure for Earth Clutter Patch model definition               **
%**                                                                          **
%**-------------------------------------------------------------------------*/
struct t_cend_clutter_patch{ /* clutter patch */
   t_spd_int8           patch_id[c_spd_name_size];      /* patch identifier */
   t_spd_int32          vertex_count;  /* number of vertices */
   t_spd_location       loc[c_cend_max_vertices]; /* location of all vertices */
   t_spd_clutter_types  clutter_type;  /* clutter type */
   t_cend_wind          wind;          /* wind information */
};
%
%/*------------------------------------------------------------------------- **
%**                                                                          **
%**    Data structure for Discrete Clutter model definition                  **
%**                                                                          **
%**-------------------------------------------------------------------------*/
struct t_cend_discrete{ /* discrete clutter */
   t_spd_int8           discrete_id[c_spd_name_size];   /* descrete identifier */
   t_spd_location       location;      /* lat & lon of discrete location */
   t_spd_discrete_types type;          /* discrete type */
   t_spd_int8           RCS_median[c_cend_max_tbls];     /* RCS profile table name */
   t_spd_int8           RCS_peak[c_cend_max_tbls];       /* RCS profile table name */
   t_spd_real           spctral_width;  /* spectral width (knots) */
   t_cend_seed          discrete_seed;          /* noise seed */
};
%#define c_cend_min_spectral_width 0.0
%#define c_cend_max_spectral_width 1.0
%#define c_cend_default_spectral_width c_cend_min_spectral_width
%
%/*------------------------------------------------------------------------- **
%**                                                                          **
%**    Data structure for Precipitation Cell model definition                **
%**                                                                          **
%**-------------------------------------------------------------------------*/
struct t_cend_precip_cell{                      /* precipitation cell model */
   t_spd_int8           cell_id[c_spd_name_size];       /* precip. cell identifier */
   t_spd_location       loc;           /* location */
   t_spd_water_state    water_state;   /* water state */
   t_spd_real           radius;        /* circular radius (km) */
   t_spd_real           low_alt;       /* lower altitude (m) */
   t_spd_real           up_alt;        /* upper altitude (m) */
   t_spd_real           water_qty;     /* water quantity */
                                       /* rain/snow: (mm/hr) */
                                       /* fog/cloud: (gm/m3) */
   t_spd_real           temperature;            /* temperature (degrees) */
   t_spd_real           wind_speed;    /* wind speed (knots) */
   t_spd_real           wind_dir;      /* wind direction rsp to North (degrees) */
};
%
%#define c_cend_min_radius 0.0
%#define c_cend_max_radius 60.0
%#define c_cend_default_radius 5.0
%#define c_cend_min_low_alt 0.0
```

```
%#define c_cend_min_up_alt 0.0
%#define c_cend_max_low_alt 8000.0
%#define c_cend_max_low_alt_fog 100.0
 #define c_cend_min_water_qty_rain 0.25
  efine c_cend_min_water_qty_snow 0.025
 #define c_cend_min_water_qty_fog 0.04
%#define c_cend_min_water_qty_cloud 0.5
%#define c_cend_max_water_qty_rain 40.0
%#define c_cend_max_water_qty_snow 4.0
%#define c_cend_max_water_qty_fog 2.3
%#define c_cend_max_water_qty_cloud 2.5
%#define c_cend_default_water_qty_rain 4.0
%#define c_cend_default_water_qty_snow 0.4
%#define c_cend_default_water_qty_fog 0.15
%#define c_cend_default_water_qty_cloud 0.8
%#define c_cend_min_temperature_snow (-40.0)
%#define c_cend_max_temperature_snow 0.0
%#define c_cend_min_temperature 0.0
%#define c_cend_max_temperature 40.0
%#define c_cend_min_wind_speed 0.0
%#define c_cend_max_wind_speed 100.0
%#define c_cend_min_wind_dir 0.0
%#define c_cend_max_wind_dir 360.0
%
%/*---------------------------------------------------------------------------- **
%**                                                                              **
%**    Data structure for Chaff Bundle definition                               **
%**                                                                              **
%**-----------------------------------------------------------------------------*/
  uct t_cend_chaff_bundle{                          /* chaff bundle */
   t_spd_int8          chaff_id[c_spd_name_size];       /* chaff identifier */
   t_spd_location      location;        /* lat, lon (degs)pair of each chaff */
   t_spd_real          altitude;        /* altitude (m) */
   t_spd_real          weight;          /* weigth (kg) */
   t_spd_real          stag_tuning;     /* stagger tuning (MHz) */
   t_spd_real          disp_effcy;      /* dispersion efficiency (%) */
   t_spd_real          wind_speed;      /* wind speed (knots) */
   t_spd_real          wind_dir;        /* wind direction rsp to North (degrees) */
   t_cend_seed         seed;            /* noise seed */
};
%#define c_cend_min_altitude 0.0
%#define c_cend_max_altitude 20000.0
%#define c_cend_default_altitude 5000.0
%#define c_cend_min_weight 0.0
%#define c_cend_max_weight 5.0
%#define c_cend_default_weight 0.5
%#define c_cend_min_stag_tuning 1.0
%#define c_cend_max_stag_tuning 500.0
%#define c_cend_default_stag_tuning_L_band 200.0
%#define c_cend_min_disp_effcy 0.0
%#define c_cend_max_disp_effcy 100.0
%#define c_cend_default_disp_effcy 50.0
%
%/*---------------------------------------------------------------------------- **
 **                                                                             **
  *    Data structure for False target definition                              **
 **                                                                             **
 **----------------------------------------------------------------------------*/
 truct t_cend_ftarget{                        /* false target */
   t_spd_real          range_offset;  /* range offset relative to jammers
                                         actual range specified at start time */
```

```
%%define c_cend_min_range_offset 0
%%define c_cend_max_range_offset 150
%%define c_cend_default_range_offset 0
%%define c_cend_min_velocity_offset (-3.0)
%%define c_cend_max_velocity_offset 3.0
%%define c_cend_default_velocity_offset 0.0
%                                                                     **
%/*-------------------------------------------------------------------**
%**                                                                   **
%**    Data structure for time definition                             **
%**                                                                   --*/
%**----------------------------------------------------------------------
struct t_cend_time{
   t_spd_int32          dwell;
   t_spd_int32          nci;
   t_spd_int32          cpi;
   t_spd_real           offset; /* offset im msec */
};
%                                                                     **
%/*-------------------------------------------------------------------**
%**                                                                   **
%**    Data structure for Jammer definition                           **
%**                                                                   --*/
%**----------------------------------------------------------------------

                                          /* jammer model */
struct t_cend_jammer{
   t_spd_int8           jammer_id[c_spd_name_size];    /* jammer identifier */
   t_spd_location       location;                 /* location */
   t_spd_real           altitude;                 /* altitude (m) */
   t_spd_real           speed;          /* speed (knots) */
   t_spd_real           heading;        /* heading (degrees) */
   t_spd_jammer_types   type;           /* jammer type */
   t_spd_real           erp;            /* ERP (dBW) */
   t_spd_real           bandwth;        /* bandwidth (GHz) barrage only */
   t_spd_real           inst_bandwth;   /* bandwidth (MHz) */
   t_spd_real           cntr_freq;      /* GHz barrage only */
   t_spd_real           mid_freq;       /* GHz swept only */
   t_spd_real           offset_freq;    /* MHz follower */
   t_spd_real           strobe_freq;    /* strobe frequency (kHz) - partial time only */
   t_spd_real           duty_cycle;     /* duty cycle (%) - partial time only */
   t_spd_real           sweep_rate;     /* swept only kHz */
   t_spd_real           sweep_band;     /* swept only GHz */
   t_spd_real           proc_delay;     /* processing delay (msec) follower only */
   t_cend_time          start_time;     /* start time (msec) relative to TOD */
   t_cend_time          stop_time;      /* stop time (msec) relative to TOD */
   t_spd_int32          ftarget_count;  /* number of false target - deception only */
   t_spd_boolean        RSS_gen;        /* flag indicating whether attributes of
                                           false targets are RSS-generated */
   t_cend_ftarget       ftarget[c_cend_max_ftargets]; /* user-entered false targets */
   t_spd_int32          deception;      /* deception method */
   t_spd_boolean        capt_jammer;    /* allows capture of spec. jammer in spec. chan. *
   t_cend_seed  jammer_seed;            /* jammer seed */
 };
 %%define c_cend_min_jammer_speed 0.0
 %%define c_cend_max_jammer_speed 500.0
 %%define c_cend_default_jammer_speed c_cend_min_jammer_speed
 %%define c_cend_default_jammer_type c_spd_barrage
 %%define c_cend_min_erp 50.0
 %%define c_cend_max_erp 120.0
 %%define c_cend_min_bandwidth 0.0
 %%define c cend_max_bandwidth 2.0
```

D-

```
%#define c_cend_min_cntr_freq 1.0
%#define c_cend_max_cntr_freq 4.0
%#define c_cend_min_mid_freq 1.0
%#define c_cend_max_mid_freq 4.0
[illegible]fine c_cend_min_offset_freq (-5.0)
[illegible]efine c_cend_max_offset_freq 5.0
%#define c_cend_min_strobe_freq 0.0
%#define c_cend_max_strobe_freq 100.0
%#define c_cend_default_strobe_freq 1.0
%#define c_cend_min_swept_bandwidth 0.0
%#define c_cend_max_swept_bandwidth 2.0
%#define c_cend_default_swept_bandwidth 1.0
%#define c_cend_min_duty_cycle 0.0
%#define c_cend_max_duty_cycle 100.0
%#define c_cend_default_duty_cycle c_cend_max_duty_cycle
%#define c_cend_min_sweep_rate 0.0
%#define c_cend_max_sweep_rate 100.0
%#define c_cend_default_sweep_rate 1.0
%#define c_cend_min_sweep_band 0.0
%#define c_cend_max_sweep_band 100.0
%#define c_cend_min_proc_delay 0.0
%#define c_cend_max_proc_delay 10.0
%#define c_cend_default_proc_delay c_cend_min_proc_delay
%#define c_cend_min_ftarget 1
%#define c_cend_max_ftarget c_cend_max_ftargets

/*------------------------------------------------------------------------- **
**                                                                          **
**    Data structure for Target definition                                  **
                                                                            **
   -------------------------------------------------------------------------*/

truct t_cend_target{                    /* target model */
  t_spd_int8          target_id[c_spd_name_size];     /* target identifier */
  t_spd_location      loc;              /* location */
  t_spd_real          altitude;                 /* altitude (m) */
  t_spd_real          speed;            /* speed (knots) */
  t_spd_real          heading;          /* heading (degrees) */
  t_spd_int32         RCS_profile;      /* table or file */
  t_spd_int8          RCS_name[c_cend_max_tbls]; /* RCS table file name */
  t_spd_real          RCS;              /*  RCS value dBm^2 */
  t_spd_real          spectral_width; /* spectral width knots */
  t_spd_boolean       capt_target;      /* allows capture of specific target */
  t_cend_seed         target_seed;              /* target seed */
;

#define c_cend_min_target_speed 0.0
#define c_cend_max_target_speed 3.0
#define c_cend_default_target_speed 0.5
#define c_cend_min_heading 0.0
#define c_cend_max_heading 360.0
#define c_cend_default_heading c_cend_min_heading
#define c_cend_min_RCS (-10.0)
#define c_cend_max_RCS 40.0
define c_cend_default_RCS 13.0
  fine c_cend_min_target_spectral_width 0.0
efine c_cend_max_target_spectral_width 4.0
define c_cend_default_target_spectral_width 0.58
/*------------------------------------------------------------------------ **
*                                                                          **
*    Data structure for Environment Definition                             **
*
```

```
                                                                        **
%**    - External Noise Definition                                      **
%**    - Background clutter Definition                                  **
%**    - Lower Atmosphere Definition                                    **
%**    - Ionosphere Definition                                          **
%**    - Earth Clutter Definition                                       **
%**         - Clutter Patches Definition                                **
%**         - Discrete Clutter Definition                               **
%**    - Volumetric Clutter Definition                                  **
%**         - Chaff Bundles Definition                                  **
%**         - Prececiptation Cells Definition                           **
%**    - Jammers Definition                                             **
%**    - Targets Definition                                             **
%**                                                                     **
%**    IMPORTANT NOTE:                                                  **
%**        Since each definition might have a large instances, for example, **
%**        number of targets and jammers can have upto 100, the environment **
%**        definition will take a large disk space while a lot of them  **
%**        is empty, if we use a fixed-length array structure definition **
%**        as in the current design.                                    **
%**                                                                     **
%**        I (stephen Tran) strongly recommend that all the definition array **
%**        should be changed into pointer to structure.  It means that the **
%**        program has to dynamically allocate memory for the number of **
%**        instances.  An easy way to do is to call "malloc" to get enough **
%**        vertual memory for the required number of instances of the   **
%**        definitions at run time, then store the dynamic address as the **
%**        pointer to structure.                                        **
%**                                                                     **
%**        For example, target array should be defined as:              **
%**            t_cend_target *target;                                   **
%**                                                                     **
%**        and use malloc to get pointer to structure.                  **
%**                                                                     **
%**            target = (t_cend_target *) malloc (                      **
%**                       target_count*sizeof(t_cend_target));          **
%**                                                                     **
%**                                                                     **
%**        Because the data definition is a variable-lenght array,      **
%**        we can utilize XDR (eXternal Data Representation) to write   **
%**        or read from file.                                           **
%**                                                                     **
%**                                                                     **
%**-------------------------------------------------------------------------*/
%**--------------------------------------------------------------------------
%
struct  t_cend_env_def{ /* Environment Definition */
   t_cend_external_noise noise;         /* external noise */
   t_cend_bckgrd_clutter bckgrd_clutter;          /* background clutter */
   t_cend_atmosphere    atmsph;                   /* lower atmosphere */
   t_cend_ionosphere    ionsph;                   /* ionosphere */

   /* earth clutter */
   t_spd_int32          patch_count;              /* number of clutter patches */
   t_cend_clutter_patch  patch[c_cend_max_patches]; /* clutter patch */
   t_cend_seed                 ec_seed;
        t_spd_int32            discrete_count;          /* number of discretes */
   t_cend_discrete      discrete[c_cend_max_discretes]; /* discretes */
        t_cend_seed                    discrete_seed;
   /* volumetric clutter */
   t_spd_int32          chaff_count;              /* number of chaff bundles */
   t_cend_chaff_bundle  chaff[c_cend_max_chaffs]; /* chaff bundles */
   t_spd_int32          cell_count;               /* number of precipitation cells */
                                                  /* precipitation cells */
```

D-

```
   /* jammer */
   t_spd_int32          jammer_count;          /* number of jammers */
   t_cend_jammer        jammer[c_cend_max_jammers]; /* jammers */

   /* target */
   t_spd_int32          target_count;          /* number of targets */
   t_cend_target        target[c_cend_max_targets]; /* targets */
  };
#ifdef RPC_HDR
%#endif
#endif
```

THIS PAGE INTENTIONALLY LEFT BLANK

D1.10 Simulation Run Definition (CSID)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*=========================================================================

   SPD - Signal Processing Demonstrator Computer System

   (C) Copyright Her Majesty the Queen in Right of Canada
   as represented by the Minister of National Defence

   File Name:       csid_intf.x - Simulation Run Definition

                    CSID is a package under the DMT [Data Managemnet] CSC
                    in the Controller Functional area. CSID specifies the
                    Simulation Run Definition contained in a data store
                    (file) in the Controller Functional area.

   Identification: [@(#)]

   SCCS Id:         @(#)csid_intf.x 1.2 93/05/11

   Responsible Organization:

                    MacDonald Dettwiler and Associates

   Purpose:         This package declares and encapsulates the Simulation Run
                    Definition in the XDR format, i.e. the Simulation Run
                    Definition file is stored in XDR format.

                    The main reason using XDR file format is to reduce the
                    disk space. As the Simulation Run Definition contains
                    many instances of DIs, NCIs and CPIs, the definition
                    is tytpically defined as a tree structure. Because XDR
                    supports variable-length array, instead defining in
                    a fixed length array with maximum number of entries
                    (maximum number of DIs, NCIs and CPIs) the tree
                    will be defined in a variable number of entries at individual
                    sub-trees (intervals).

                    This file is written in RPC data description language in XDR
                    (eXternal Data Representation) standard. This file will use
                    "rpcgen" utility to generate the following two files:

                    csid_intf.h     - global include file for programs reading
                                      writing the Simulation Run Definition
                                      data file to include;
                    csid_intf_xdr.c - XDR source file which contains required XDR
                                      conversion routines to support the I/O
                                      service functions in csid_io.c.

   Dependency:      cdsi_intf.x - declaring the Controller/DSP message interface.

                    This package uses the data structures defined for simulation
                    run processing parameters which the Controller sends
                    to the DSP.

   Designer:        Stephen Tran

   Change History:

       SCR    Date           Author    Change made
       ---    -----------    ------    ----------------------------------------
       n/a    23-Feb-1993     cst      Initial Release
              03-Mar-1993     cst      after 40% review
                                       - removed SRSD_pathname, sim_pathname,
                                         run_ident.
                                       - changed t_cdsi_RSP_proc to t_cdsi_radar_proc
              06-May-1993     cst      include pathname to the associated Sat/Rad
                                       Defintion file by adding simdef_pathname into
                                       t_csid_run_def (Design Issue #22)
                                       this associated Sat/Rad Definition file is used
                                       for external dependency check against the
                                       Simulation Run Definition.
```

```

==========================================================================*/

#ifdef RPC_HDR
%#ifndef csid_intf_h
%#define csid_intf_h
%#include "spd_intf.h"
%#include "cdsi_intf.h"
#endif

%/*-----------------------------------------------------------------------
% *
% *     DEFINE DATA STRUCTUES
% *
% *----------------------------------------------------------------------*/
%
%/* structure for a CPI element */

struct t_csid_CPI_elm {                  /* Data elements changed every CPI */

   t_spd_boolean        capture_CPI_enable;
                                         /* indicating whether to capture
                                            this CPI for all products at burst
                                            level: PC, DPCA and all ESP products.
                                            */
};
%
%/* structure for a NCI element */

struct t_csid_NCI_elm {                  /* Data elements changed every NCI */

   /* CPI interval is defined as variable-length array data in XDR */
   t_csid_CPI_elm       CPI<c_spd_max_CPI>;
                                         /* a list of CPI bursts transmitted in
                                            the current NCI. It is defined as
                                            an array pointer and this array
                                            contains upto c_spd_max_CPI entries.

                                            Coherent bursts per NCI transmitted
                                            will be shown as the array length
                                            CPI_len in the .h file.
                                            (N_cpi_b)
                                          */
   t_spd_int32          num_bursts_integ; /* N_bursts_b */
                                         /* RUN HMI 2.2.9.1 */
                                         /* number of bursts integrated
                                            Range: 1 to number of coherent bursts
                                            per NCI */
   t_spd_boolean        capture_NCI_enable; /* RUN HMI 2.3.3 */
                                         /* indicating whether to capture
                                            this NCI for NCI interval products:
                                            NCI integrated outputs. */
};
%
%/* data structure for RAR paramters */

struct t_csid_RAR {      /* RAR parameters */

   t_spd_int32          num_PRF_used;    /* RAR_prfs */
                                         /* number of PRFs to group in range
                                            ambiguity resolution
                                            Range: 1 to number of NCIs */
   t_spd_real           footprint_len;   /* fpl */
                                         /* footprint length used for two or
                                            three PRF RAR (km) */
};
%
%/* data structure for a Dwell Interval element */

struct t_csid_DI_elm {            /* Data elements changed every DI */
```

```
     /* NCI interval is defined as variable-length array data */
    t_csid_NCI_elm        NCI<c_spd_max_NCI>;
                                          /* a list of Non-coherent Interval in
                                             the current dwell interval. It is
                                             defined as an array pointer and
                                             this array contains upto
                                             c_spd_max_NCI entries.
                                             NCI intervals per DI is shown
                                             as the array length (NCI_len)
                                             in the .h file */
    t_spd_int32           num_pulses_proc;/* N_pp */
                                          /* number of pulses processed (centred
                                             about beam centre range */
    t_cdsi_pulse_doppler PD;              /* pulse dopper */
    t_cdsi_ASAR_DPCA      ASAR_DPCA;      /* ASAR DPCA */
    t_spd_int32           OS_NCI_rank;    /* os_rank */
                                          /* OS rank
                                             Rank=1 means take min. value;
                                             rank=2 is to take second smallest,
                                             etc.
                                             Range: 1 to number bursts
                                             integrated */
    /* CFAR */
    t_cdsi_detect_thrshd detect_thrshd;  /* detection thresholding parameters */

    t_csid_RAR           parm_est;       /* parameter estimation */

    t_spd_boolean        capture_DI_enable;
                                         /* indicating whether to capture
                                            this DI for DI interval products:
                                            pulse compression MF. */
 };
 %/*
 % * data structure for simulation run definition. Each definition contains
 % * - a list of dwell interval
 % * - general configuration
 % * - ECCM configuration and processing parameters
 % * - radar processing parameters
 % * - Intermediate products capture options
 % * - Signal data reduction options
 % *
 % */
 struct t_csid_run_def { /* Simulation Run definition */
    /* hard linkage to the associated Simulation Run Definition */
    char simdef_pathname[c_spd_path_size];/* full directory and file name for
                                             Simulation Run Definition */

    /* Dwell interval is defined as variable-length array data */
    t_csid_DI_elm        DI<c_spd_max_DI>;
                                         /* a list of dwell intervals in the
                                            current run; it is defined as
                                            an array pointer and this array
                                            contains upto c_spd_max_DI entries */

    t_cdsi_general_config general_config:/* general configuration */
    t_cdsi_ECCM_config   ECCM_config;    /* ECCM configuration */
    t_cdsi_ECCM_proc     ECCM_proc;      /* ECCM processing parameters */

    t_cdsi_radar_proc    radar_proc;     /* radar processing parameters */
    t_cdsi_IR_capture    IR;             /* Intermediate Products capture
                                            options */
    t_cdsi_signal_reduction sig_reduct;  /* signal data reduction options */
 };

 #ifdef RPC_HDR
 %
 %
 %/*___________________________________________________________________________
 % *
```

```
% *  SERVICE FUNCTIONS SPECIFICATION
% *
% *---------------------------------------------------------------------------*/
%/*============================================================================
%
%   Function:   f_csid_read
%
%   Description:
%      This function reads a Simulation Run Definition data file
%      named by pathname into in-memory data structure.
%      pathname points to a character string that contains the directory and
%      the name of Simulation Run generation definition file to be read.
%      The memory for the data structure will be allocated by this function.
%      Therefore, it is the caller's responsibility to release the memory.
%
%   Arguments:
%      pathname                  - in    - directory and file name of
%                                          the Simulation Run
%                                          Definition data log.
%      run_def                   -   out - Simulation Run Definition
%
%   External variables:
%      none
%
%   Exceptions:
%      s_spd_failure returned if the provided pathname is not found or
%      can not be read.
%
%   Notes:
%      none
% ===========================================================================*/
%extern t_spd_status f_csid_read (
%               char *pathname,
%               t_csid_run_def *run_def);
%
%
%/*============================================================================
%
%   Function:   f_csid_write
%
%   Description:
%      This function writes a Simulation Run Definition data
%      structure to a data file named by pathname.
%      pathname points to a character string that contains the directory and
%      the name of Simulation Run generation definition file to be write.
%
%   Arguments:
%      pathname                  - in    - directory and file name of
%                                          the Simulation Run
%                                          Definition data log.
%      run_def                   - in    - Simulation Run Definition
%
%   External variables:
%      none
%
%   Exceptions:
%      s_spd_failure returned if the data can not be written to the provided
%      pathname.
%
%   Notes:
%      none
%
%============================================================================*/
%extern t_spd_status f_csid_write (
%               char *pathname,
%               t_csid_run_def *run_def);
#endif

#ifdef RPC_HDR
%#endif
```

```
#endif
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csid_io.c - Simulation Run Definition I/O functions

Identification: [@(#)]

SCCS Id:        @(#)csid_io.c   1.1 93/04/22

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package implements read/write functions for the
                Simulation Run Definition data store.

Designer:       Stephen Tran

Change History:

   SCR  Date         Author   Change made
   ---  -----------  ------   -----------------------------------------
   n/a  29-Jan-1993  cst      Initial Release

=========================================================================*/

/* Included external declarations */
#include <stdio.h>
#include <rpc/rpc.h>
#include "spd_intf.h"
#include "cerr_intf.h"
#include "csid_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* Implementation */
/*=========================================================================

   Function:    f_csid_read

   Description:
      This function reads a Simulation Run Definition data file
      named by pathname into in-memory data structure.
      pathname points to a character string that contains the directory and
      the name of Simulation Run generation definition file to be read.
      The memory for the data structure will be allocated by this function.
      Therefore, it is the caller's responsibility to release the memory.

   Arguments:
      pathname                  - in    - directory and file name of
                                          the Simulation Run
                                          Definition data log.
      run_def                   -  out  - Simulation Run Definition

   External variables:
      none

   Exceptions:
      s_spd_failure returned if the provided pathname is not found or
      can not be read.

   Notes:
      none
```

```

    High Level PDL
       BEGIN  f_csid_read
         USE fopen TO open Simulation Run Definition file
         with "read"-option
         IF file does not exist or can not be read THEN
            return s_spd_failure
         ELSE
            USE xdrstdio_create TO initialize a record-oriented XDR stream
            object (xdrs) in XDR_DECODE mode, which will be used in the
            subsquent XDR calls
            USE xdr_t_csid__run_def TO read the Simulation Run Definition
            file in XDR format into in-memory data structure
            IF error THEN return s_spd_failure
            USE xdr_destroy TO destroy xdrs
         ENDIF
       END f_csid_read
========================================================================*/
t_spd_status f_csid_read (
                char *pathname,
                t_csid_run_def *run_def)
{
}


/*=======================================================================

   Function:    f_csid_write

   Description:
      This function writes a Simulation Run Definition data
      structure to a data file named by pathname.
      pathname points to a character string that contains the directory and
      the name of Simulation Run generation definition file to be write.

   Arguments:
      pathname                  - in    - directory and file name of
                                          the Simulation Run
                                          Definition data log.
      run_def                   - in    - Simulation Run Definition

   External variables:
      none

   Exceptions:
      s_spd_failure returned if the data can not be written to the provided
      pathname.

   Notes:
      none

   High Level PDL
      BEGIN  f_csid_write
        USE fopen TO open Simulation Run Definition file
        with "write" option
        IF file does not exist or can not be write THEN
           return s_spd_failure
        ELSE
           USE xdrstdio_create TO initialize a record-oriented XDR stream
           object (xdrs) in XDR_ENCODE, which will be used in the subsquent
           XDR calls
           USE xdr_t_csid__run_def TO write the Simulation Run Definition
           structure to data file in XDR format.
           IF error THEN return s_spd_failure
           USE xdr_destroy TO destroy xdrs
        ENDIF
      END f_csid_write
========================================================================*/
t_spd_status f_csid_write (
                char *pathname,
                    t_csid_run_def *run_def)
{
}
```

D1.11 Simulation Products (CSIP)

The source file of this CSU (csip_intf.x) is provided in Appendix A of the DSP/Testbed Interface Detailed Design document [R-1].

THIS PAGE INTENTIONALLY LEFT BLANK

D1.12 Signal Generation Products (CGEP)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/** Identification cgep_intf1.x 1.4 93/08/16 15:47:25 */

/*===========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      cgep_intf1.x - RSS Signal Generation Intermediate products

Identification: [@(#)]

SCCS Id:        @(#)cgep_intf1.x        1.1 93/04/27

Responsible Organization:

                Raytheon Canada Limited

Purpose:        This header file declares variables and data structures
                for the XDR-formatted intermediate product files generated
                by the RSS during a Signal Generation run.

                This file is written in RPC data description language in XDR
                (eXternal Data Representation) standard. This file will use
                "rpcgen" utility to generate the following two files:

                cgep_intf1.h      - global include file for both tasks
                cgep_intf1_xdr.c  - XDR source file which contains required XDR
                                    conversion routines to support the RPC
                                    message data types.

Designer:       Marg Baskin

Change History:

   SCR    Date        Author    Change made
   ---  -----------   ------  ------------------------------------------
   n/a  08-APR-1993    mdb    Initial Release

===========================================================================*/
#ifdef RPC_HDR
%#ifndef cgep_intf1_h
%#define cgep_intf1_h
%#include "spd_intf.h"
%#include "crgi_intf.h"
%#include "sgen_intf.h"
#endif
%/****************************************************************************
%
%                intermediate product file id
%
% ****************************************************************************/
enum t_cgep_filetype {
     c_cgep_rss_clut,
     c_cgep_rss_exn,
     c_cgep_rss_jam,
     c_cgep_rss_tar,
     c_cgep_rss_comp,
     c_cgep_rss_post_nl,
     c_cgep_rss_if_filt,
     c_cgep_rss_post_sd,
     c_cgep_rss_ant_patrn,
        c_cgep_rss_filt_rsp,
        c_cgep_rss_post_ad
     };

%/***********************************************************************
%
%   Element type indicates the type of samples stored in the file
%    (complexS, real, integer, etc.
%
% **********************************************************************/

enum t_cgep_element_type {
   c_cgep_float_elm,           /* floating-point element */
   c_cgep_double_elm,          /* double precision floating-point element */
   c_cgep_complexS_elm,        /* single precision complex element */
   c_cgep_complexD_elm,        /* double precision complex element */
   c_cgep_int16_elm,           /* short integer element */
   c_cgep_int32_elm           /* long integer element */
};

%/***********************************************************************
%
%    Data type for intermediate products
%    (complexS, real, integer, etc.
%
% **********************************************************************/

union t_cgep_data       switch( t_cgep_element_type     element_type )
{
   case c_cgep_complexS_elm:               /* complex single precision variable */
        t_spd_complexS          data_complexS<c_sgen_max_int_prod_data>;

   case c_cgep_int16_elm:                  /* short integer variable */
        t_spd_int16             data_int16<c_sgen_max_int_prod_data>;

   default:
        void;

};

struct t_cgep_control_block              /* CPI header control block */
{
    t_spd_int32                  cpi_number;       /* number of the current CPI */
    t_spd_card32                 number_of_samples;
    t_spd_real                   prf;
    u_int                        data_offset;          /* offset in bytes from beginnin
                                                  file to the data record */
};

struct t_cgep_header_block                /* intermediate product header block */
{
    t_cgep_filetype              filetype;         /* ID of intermediate product file
    char                         satdef_pathname[c_spd_path_size];
    char                         envdef_pathname[c_spd_path_size];
    t_cgep_element_type          elm_type;
    u_int                        elm_size;     /* element size in bytes. It is the size
                                         of one element in a data record,
                                         which varies depending on the element
                                         type */


    t_cgep_control_block         control_block<c_sgen_max_cpis>;
 };
```

```
typedef t_spd_complexS                          t_cgep_if_filter_chan[c_sgen_nsegif];
                                                       /* IF Filter response vector for a single channel

 struct t_cgep_if_filter_header                        /* IF Filter file header block  */
 {
     char                                       satdef_pathname[c_spd_path_size];
     char                                       envdef_pathname[c_spd_path_size];
     t_cgep_element_type                        elm_type;
     u_int                                      elm_size;     /* element size in bytes. */
     t_spd_card32                               nsegif;
 };

 struct t_cgep_if_filter                               /* IF Filter response vector output file */
 {
     t_cgep_if_filter_header                    if_filter_header;
     t_cgep_if_filter_chan                      if_filter_chan<c_sgen_max_antennas>;
   };

 enum t_cgep_ant_type {
    c_cgep_RRA_ant,
    c_cgep_CFA_ant
 };

 struct t_cgep_ant_patrn_cntrl_block                   /* antenna pattern intermediate product control
 {
    t_spd_int32                                 cpi_number;
    u_int                                       data_block_offset;
 };

 struct t_cgep_ant_patrn_header                        /* antenna pattern intermediate product header b
 {
    t_cgep_filetype                             filetype;
    char                                        satdef_pathname[c_spd_path_size];
    char                                        envdef_pathname[c_spd_path_size];
    t_cgep_ant_patrn_cntrl_block                control_block<c_sgen_max_cpis>;
 };

 struct t_cgep_ant_gain_table
 {
    t_spd_real                                  indep<c_sgen_gain_array_size>;
    t_spd_complexS                              dep<c_sgen_gain_array_size>;
 };

 struct t_cgep_CFA_ant_gains
 {
    t_cgep_ant_gain_table                       gain_u;
    t_cgep_ant_gain_table                       gain_v;
    t_cgep_ant_gain_table                       e_gain_u;
    t_cgep_ant_gain_table                       e_gain_v;
 };

 struct t_cgep_CFA_ant_db
 {
    t_cgep_CFA_ant_gains                        tx<1>;
    t_cgep_CFA_ant_gains                        rx<c_spd_max_tot_ph_ctrs>;
 };

 struct t_cgep_RRA_ant_patrn
 {
    t_spd_complexS                              tot_pat<c_sgen_nu_2>;
    t_spd_complexS                              epat_u<c_sgen_nu>;
    t_spd_complexS                              epat_v<c_sgen_nu>;
 };
```

```
struct t_cgep_RRA_ant_index
{
   t_spd_real                     u0;
   t_spd_real                     v0;
   t_spd_real                     recip_du;
};

struct t_cgep_RRA_ant_db
{
   t_cgep_RRA_ant_patrn           tx;
   t_cgep_RRA_ant_patrn           rx;
   t_cgep_RRA_ant_index           index;
};

union t_cgep_main_ant switch (enum t_cgep_ant_type ant_type)
{
      case c_cgep_RRA_ant :
         t_cgep_RRA_ant_db         RRA<1>;

      case c_cgep_CFA_ant :
         t_cgep_CFA_ant_db         CFA;
};

struct t_cgep_ant_dbase
{
   t_cgep_main_ant                main_ant;
   t_cgep_CFA_ant_gains           aux_ant<c_sgen_max_channels>;
};

#ifdef RPC_HDR
%t_spd_status f_cgep1_write (t_sgen_cgep_write_flag flag,
%                            t_cgep_filetype        filetype,
%                            t_crgi_gen_parms       *genparms,
%                            t_sgen_number_of_cpis  number_of_cpis,
%                            t_spd_int32            cpi_number,
%                            t_spd_card32           number_of_samples,
%                            t_spd_real             prf,
%                            t_spd_boolean          magnitude_sqrd,
%                            t_cgep_data            *array,
%                            t_sutl_error_num       *error_num);
%
%
#endif

#ifdef RPC_HDR
%#endif
#endif
```

```
/*=====================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      cgep_intf2.x  - RSS derived processing parameters
                                XDR file

                CGEP is a package under the DMT [Data ManagemenT] CSC
                in the Controller functional area. CGEP specifies how
                the RSS products (RSS Intermediate Products and RSS
                derived processing parameters) are stored in a UNIX
                external file.

                The RSS derived processing parameters file format is described
                in this file, but The RSS Intermediate Products file format
                will be decribed in a separate XDR file (see cgep_intf1.x).

Identification: [@(#)]

SCCS Id:        @(#)cgep_intf2.x        1.4 93/09/09

Responsible Organization:

                MacDonald Dettwiler & Associates

Purpose:        This package declares constants, data types for the processing
                parameters derived from RSS.

                This file is written in RPC data description language in XDR
                (eXternal Data Representation) standard. This file will use
                "rpcgen" utility to generate the following two files:

                cgep_intf2.h    - global include file;
                cgep_intf2_xdr.c- XDR source file which contains required XDR
                                  conversion routines to support read/write
Designer:       Stephen Tran      these output files generated by the RSS.

Change History:

   SCR   Date          Author   Change made
   ---   -----------   ------   ------------------------------------------
   n/a   20-Apr-1993   cst      Initial Release
         06-May-1993   cst      update for Design Issue #22
                                - removed all parameters which are defined
                                  in Sat/Rad Definition
                                - added linkage to the Sat/Rad Definition
                                  file which was used to execute the generation
                                  run
                                - added tx_waveform_dur
         21-Jul-1993   cst      cleanup and added I/O function spec.
         30-Aug-1993   cst      added free function

=====================================================================*/

#ifdef RPC_HDR
%#ifndef cgep_intf2_h
%#define cgep_intf2_h
%#include "spd_intf.h"
#endif
%
%/* structure for a CPI element */

struct t_cgep_CPI_elm {                    /* Data elements changed every CPI */

   t_spd_real             platfm_rad_vel; /* V-p          RSS derived */
                                           /* platform radial speed along
```

```
                                          boresight vector (m/s) */

    t_spd_real            earth_rad_vel;  /* V_e          RSS derived */
                                          /* earth radial speed along
                                             boresight vector (m/s) */

    t_spd_int32           num_samp_input; /* N_sg         RSS derived */
                                          /* number of samples per burst in
                                             input data stream */

    t_spd_real            rx_tm_start_burst;
                                          /* t_rx_b    RSS derived */
                                          /* receive time for start of burst
                                             relative to beginning of first
                                             burst on the first phase centre
                                             (sec) */
    t_spd_real            CPI_samplg_ofst;/* tau_so_b    RSS derived */
                                          /* time interval between first
                                             received sample in a burst and
                                             the leading edge of that burst
                                             (sec) */

};
%
%/* structure for a NCI element */

struct t_cgep_NCI_elm {                   /* Data elements changed every NCI */

    /* CPI interval is defined as variable-length array data in XDR */
    t_cgep_CPI_elm        CPI<c_spd_max_CPI>;
                                          /* a list of CPI bursts transmitted in
                                             the current NCI. It is defined as
                                             an array pointer and this array
                                             contains upto c_spd_max_CPI entries.

                                             Coherent bursts per NCI transmitted
                                             will be shown as the array length
                                             CPI_len in the .h file.
                                             (N_cpi_b) */

    t_spd_real            inter_ph_ctr_sep;
                                          /* dsep_b       RSS derived */
                                          /* inter-phase centre separation
                                             (metres) */

};
%
%/* data structure for a Dwell Interval element */

struct t_cgep_DI_elm {            /* Data elements changed every DI */

    /* NCI interval is defined as variable-length array data */
    t_cgep_NCI_elm        NCI<c_spd_max_NCI>;
                                          /* a list of Non-coherent Interval in
                                             the current dwell interval. It is
                                             defined as an array pointer and
                                             this array contains upto
                                             c_spd_max_NCI entries.
                                             NCI intervals per DI is shown
                                             as the array length (NCI_len)
                                             in the .h file */

    /* engagement geometry */
    t_spd_real            platfm_heading; /* platform heading wrt NED system
                                             (radian)             RSS derived */
                                                                  RSS derived */
    t_spd_real            look_depression; /* Theta
                                          /* look vector depression angle */
    t_spd_real            look_azimuth;   /* Phi          RSS derived */
                                          /* look vector azimuth anlge (radian) */
    t_spd_real            ref_range;      /* R_ref                RSS derived */
                                          /* burst reference range corresponding
                                             to range bin zero (km) */


    /* receive front end */
    t_spd_real            noise_power_main;/* P_n          RSS derived */
    t_spd_real            noise_power_aux; /* P_n_a        RSS derived */
                                          /* noise power referenced at the A/D
```

```
                                           converter output (unit: A/D voltage
                                           resolution squared) */
   /* waveform */
   t_spd_complexS       pulse_replica[c_spd_replica_size];
                                        /* rep_dc(r)    RSS derived */
                                        /* vector of time-domain samples
                                           (transmitted order) of zero Doppler
                                           range pulse replica.
                                           align actual replica in {0,..,N-1}
                                           indices. */
   t_spd_int32          replica_len;    /* N_tau        RSS derived */
                                        /* number of samples in time-domain
                                           replica.
                                           Range: 1 to c_spd_replica_size */

   /* parameter estimation */
   t_spd_real           RCS_scale_factor;
                                        /* rcs_scale    RSS derived */
                                        /* conversion factor used to convert
                                           from digital power units to
                                           metres-squared RCS units */
};
%
%/* data structure for RSS derived processing parameters */

struct t_cgep_RSS_process_parms { /* RSS derived processing parameters */

   /* Dwell interval is defined as variable-length array data */
   t_cgep_DI_elm        DI<c_spd_max_DI>;
                                        /* a list of dwell intervals in the
                                           current run; it is defined as
                                           an array pointer and this array
                                           contains upto c_spd_max_DI entries */

   /* header information */
   char SRSD_pathname[c_spd_path_size]; /* full directory and file name for
                                           the SRSD file, which is a pair of
                                           the RSS derived processing parameters
                                           file. Both files are the output files
                                           of a signal generation with the same
                                           gen_ident provided by the user. */
                                        /* NOTE: both SRSD file and RSS derived
                                           processing parameters file will have
                                           gen_ident as file subfix. */
   char satdef_pathname[c_spd_path_size];/* full directory and file name for
                                           the associated Satelleite/Radar
                                           Definition */
                                         /* this file is the duplicated file of
                                            the Sat/Rad Definition file which
                                            is selected by the user during a
                                            signal generation activity */
   char envdef_pathname[c_spd_path_size];/* full directory and file name for
                                           Environment Definition which is
                                           selected for signal generation */

   t_spd_real           tx_waveform_dur;/* RSS derived */
                                        /* transmit waveform duration (sec) */
                                        /* an absolute time duration for
                                           the signal data file which is used
                                           to compute the real time rate
                                           factor in the simulation run */

};

#ifdef RPC_HDR
%
%
%/*-----------------------------------------------------------------
% *
% *   SERVICE FUNCTIONS SPECIFICATION
% *
% *-------------------------------------------------------------------------*/
%/*==========================================================================
%
%   Function:   f_cgep_read
%
%   Description:
%      This function reads a RSS derived processing parameters data file
%      named by pathname into in-memory data structure.
%      pathname points to a character string that contains the directory and
%      the name of RSS derived processing parameters file to be read.
%
%   Arguments:
%      pathname                 - in    - directory and file name of
%                                         the Simulation Run
%                                         Definition data log.
%      RSS_process_parms        -   out - RSS derived processing parameters
%                                         structure
%
%   External variables:
%      none
%
%   Exceptions:
%      s_spd_failure returned if the provided pathname is not found or
%      can not be read.
%
%   Notes:
%      The definition structure contains a starting pointer to a tree structure,
%      which accomodates all time sub-intervals (all dwells, all NCIs within a
%      dwell, all CPIs within a NCI).  The starting pointer DI_val points to
%      dwells, then the pointer NCI_val within the dwell structure points to
%      NCIs and the pointer CPI_val within the NCI structure points to CPIs.
%
%      The memory for the data structure of all time sub-intervals (Dwells,
%      NCIs and CPIs) will be allocated by this function using XDR functions.
%      After this function is called, it is the caller's responsibility to
%      release the tree structure memory by calling f_cgep_free function
%
% ==========================================================================*/
%extern t_spd_status f_cgep_read (
%#ifdef __STDC__
%                    char *pathname,
%                    t_cgep_RSS_process_parms *RSS_process_parms
%#endif
%);
%
%/*==========================================================================
%
%   Function:   f_cgep_write
%
%   Description:
%      This function writes a RSS derived processing parameters data
%      structure to a data file named by pathname.
%      pathname points to a character string that contains the directory and
%      the name of RSS derived processing parameters file to be write.
%
%   Arguments:
%      pathname                 - in    - directory and file name of
%                                         the Simulation Run
%                                         Definition data log.
%      RSS_process_parms        - in    - RSS derived processing parameters
%                                         structure
%
%   External variables:
%      none
%
%   Exceptions:
%      s_spd_failure returned if the data can not be written to the provided
%      pathname.
%
```

```
%   Notes:
%       none
%
%=============================================================================*/
%extern t_spd_status f_cgep_write (
%#ifdef __STDC__
%                   char *pathname,
%                   t_cgep_RSS_process_parms *RSS_process_parms
%#endif
%);
%
%/*=============================================================================
%
%   Function:   f_cgep_free
%
%   Description:
%      This function frees the memory of all time sub-intervals pointed by the
%      RSS derived processing parameters data structure.
%      All dwells, NCIs and CPIs memory under this RSS derived processing
%      parametes structure will be de-allocated, but the RSS derived processing
%      parameter structure itself (parameters at run level) will not.  The
%      pointer DI_val will be reset to NULL.
%
%   Arguments:
%      RSS_process_parms        -    in  - RSS derived processing parameters
%
%   External variables:
%      none
%
%   Exceptions:
%     none
%
%   Notes:
%      none
%
% ============================================================================*/
%extern void f_cgep_free (
%#ifdef __STDC__
%                   t_cgep_RSS_process_parms *RSS_process_parms
%#endif
%);

#endif

#ifdef RPC_HDR
%#endif
#endif
```

THIS PAGE INTENTIONALLY LEFT BLANK

D1.13 Control Generation HMI (CGEH)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      cgeh_intf.h - Controller Generation HMI External Include file

Identification: [@(#)]

SCCS Id:        @(#)cgeh_intf.h 1.4 93/06/09

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file declares funtion description for communicating
                with the Monitor/Control Generation task (CGEH task) through
                X window.

Designer:       Stephen Tran

Change History:

     SCR    Date         Author   Change made
     ---    -----------  ------   ----------------------------------------
     n/a    16-Feb-1993   cst     Initial Release
            19-May-1993   cst     removed c_cgeh_gen_denied, c_cgeh_abort_denied
                                  from t_cgeh_comm_msg_types.
                                  (now these responses are returned via RPC
                                   sendreply instead of X).

=========================================================================*/

#ifndef cgeh_intf_h
#define cgeh_intf_h
#include <X11/Intrinsic.h>

/*-----------------------------------------------------------------------*
 *                                                                       *
 *    DEFINE CONSTANTS                                                   *
 *                                                                       *
 *-----------------------------------------------------------------------*/
#define c_cgeh_comm_data_name   "cgeh_data_"
#define c_cgeh_comm_type_name   "cgeh_type_"


/*-----------------------------------------------------------------------*
 *                                                                       *
 *    DEFINE ENUMERATED DATA TYPES                                       *
 *                                                                       *
 *-----------------------------------------------------------------------*/
/*
   the message type sent from Generation Server to Monitor/Control
   Generation HMI task.
 */

typedef enum {
    c_cgeh_gen_compl,            /* generation completed message */
    c_cgeh_gen_failed,           /* generation failed message */
    c_cgeh_abort_compl,          /* abort completed message */
    c_cgeh_abort_failed,         /* abort failed message */
    c_cgeh_in_progress           /* generation progress message */
} t_cgeh_comm_msg_types;


/*-----------------------------------------------------------------------*
 *                                                                       *
 *    DEFINE DATA STRUTURES                                              *
 *                                                                      *
 * ---------------------------------------------------------------------*/
/*
    Data structure of the Generation Status (start/abort) sent from
    the Generation Server task to the Monitor/Control Generation task
    via X property.

    Source:     Controller Generation Server task
    Sink:       Controller Monitor/Control Generation HMI task

    This structure is used in Dialogue intertask communication using X Property.

        msg_type           tag indicating message type or event occurred
        percent            percent complete
                           Note: the percent attribute is only valid for
                                 the following message types:
                                     c_cgeh_gen_compl,
                                     c_cgeh_gen_failed,
                                     c_cgeh_in_progress
                                 and has no meaning for the abort message
                                 types:
                                     c_cgeh_abort_compl,
                                     c_cgeh_abort_failed

 */
typedef struct {
    t_cgeh_comm_msg_types       msg_type;       /* message type */
    t_spd_real                  percent;        /* percent complete */
} t_cgeh_comm_data;

/*
   Intertask communication identifier
 */
typedef struct {
    Display *display;        /* connection to the X server              */
    Atom    data_atom;       /* atom representing property name         */
    Atom    data_type_atom;  /* atom representing data type in property */
} t_cgeh_comm_ITC_id;

/*--------------------------------------------------------------------- *
 *                                                                      *
 *   DEFINE FUNCTION SPECIFICATION                                      *
 *                                                                      *
 *----------------------------------------------------------------------*/

/*======================================================================

    Function:    f_cgeh_comm_initialize

    Description:
       This function creates X atoms (data atom and data type atom)
       using appended pid (process id of the monitor/Control Generation
       HMI task) and returns an intertask communication identifier (ITC_id).
       The returned ITC_id will be used in the subsquent calls in
       f_cgeh_comm_put and f_cgeh_comm_get.

    Arguments:
       display                  - in    - X Display structure; returned from
                                          XOpenDisplay
       pid                      - in    - pid of Monitor/Control Generation
                                          HMI task
       ITC_id                   -   out - intertask communication identifier

    Exceptions:
       s_spd_failure returned if it failed to create atoms

    Note:
       This must be called by both Monitor/Control Generation HMI task
       and Controller Generataion server.  The pid of the Monitor/Control
       Generation must be known by the Controller Generation Server
```

```
        in order to establish the x communication.

  ===========================================================================*/
extern t_spd_status f_cgeh_comm_initialize (
                    Display *display,
                    t_spd_int32 pid,
                    t_cgeh_comm_ITC_id *ITC_id);

/*===========================================================================

    Function:    f_cgeh_comm_get

    Description:
        This function retrieves communication data from root window's property.

    Arguments:
        ITC_id           - in    - Intertask communication identifier
        comm_data        -   out - communication data

    Exceptions:
        s_spd_failure returned if it failed to read root window property.

  ===========================================================================*/
extern t_spd_status f_cgeh_comm_get (
                    t_cgeh_comm_ITC_id      *ITC_id,
                    t_cgeh_comm_data        *comm_data);

/*===========================================================================

    Function:    f_cgeh_comm_put

    Description:
        This function stores communication data into root window's property
        and then send them to the X server.

    Arguments:
        ITC_id           - in    - Intertask communication identifier
        comm_data        - in    - communication data

    Exceptions:
        none

  ===========================================================================*/
extern t_spd_status f_cgeh_comm_put (
                    t_cgeh_comm_ITC_id      *ITC_id,
                    t_cgeh_comm_data        *comm_data);


/*===========================================================================

    Function:    f_cgeh_comm_is_wanted

    Description:
        This function checks for property change events on the root window
        and determines whether the event (the message) is sent by
        the Generation Server is for him (the Monitor/Control Generation HMI).

    Arguments:
        ITC_id                     - in    - Intertask communication identifier
        event                      - in    - X event

    Exceptions:
        none

  ===========================================================================*/
extern t_spd_boolean f_cgeh_comm_is_wanted (
                    t_cgeh_comm_ITC_id      *ITC_id,
                    XEvent  *event);

#endif  /* end of cgeh_intf_h */
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      cgeh_comm.c - Intertask communication from
                              Controller Generation server task to
                              Controller Monitor/Control Generation HMI task.

Identification: [@(#)]

SCCS Id:        @(#)cgeh_comm.c 1.2 93/05/10

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file implements the intertask communication service
                functions using X property to facilitate communication
                from Controller Generation Server task to Controller
                Monitor/Control Generation HMI task which is waiting for
                X events.

Designer:       Stephen Tran

Change History:

   SCR  Date        Author  Change made
   ---  ----------- ------  ------------------------------------
   ??? 09-Mar-1993  cst    Initial Release

==========================================================================*/

/* Included external declarations */
#include <X11/Xlib.h>
#include "spd_intf.h"
#include "cgeh_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* common constants */
/* none */

/* Implementation */
/*==========================================================================

   Function:    f_cgeh_comm_initialize

   Description:
      This function creates X atoms (data atom and data type atom)
      using appended pid (process id of the monitor/Control Generation
      HMI task) and returns an intertask communication identifier (ITC_id).
      The returned ITC_id will be used in the subsquent calls in
      f_cgeh_comm_put and f_cgeh_comm_get.

   Arguments:
      display                 - in    - X Display structure; returned from
                                        XOpenDisplay
      pid                     - in    - pid of Monitor/Control Generation
                                        HMI task
      ITC_id                  -   out - intertask communication identifier

   Exceptions:
      s_spd_failure returned if it failed to create atoms

```

```
     Note:
        This must be called by both Monitor/Control Generation HMI task
        and Controller Generataion server.  The pid of the Monitor/Control
        Generation must be known by the Controller Generation Server
        in order to establish the x communication.

      High Level PDL
         BEGIN f_cgeh_comm_initialize
            USE XInternAtom() TO create data atom with name
                (c_cgeh_comm_data_name + pid string)
            USE XInternAtom() TO create data type atom with name
                (c_cgeh_comm_type_name + pid string)
            IF atom creation is successful THEN
               store display, data atom, data type atom to ITC_id
               return s_spd_success
            ELSE
               return s_spd_failure
            ENDIF
         END f_cgeh_comm_initialize
      END of PDL
   ============================================================================*/
t_spd_status f_cgeh_comm_initialize (
                  Display *display,
                  t_spd_int32 pid,
                  t_cgeh_comm_ITC_id *ITC_id)
{
};

/*============================================================================

     Function:    f_cgeh_comm_get

     Description:
        This function retrieves communication data from root window's property.

     Arguments:
        ITC_id           - in    - Intertask communication identifier
        comm_data        -   out - communication data

     Exceptions:
        s_spd_failure returned if it failed to read root window property.

      High Level PDL
         BEGIN f_cgeh_comm_get
            USE XGetWindowProperty() TO retrieve comm data from root window's
                property
            IF retrieval is successful THEN
                copy comm data to comm_data
                USE XFree() TO free data storage
                return s_spd_success
            else
                return s_spd_failure
            end if
         END f_cgeh_comm_get
      END of PDL
 ============================================================================*/
t_spd_status f_cgeh_comm_get (
                  t_cgeh_comm_ITC_id      *ITC_id,
                  t_cgeh_comm_data        *comm_data)
{
};

/*============================================================================

     Function:    f_cgeh_comm_put

     Description:
        This function stores communication data into root window's property
        and then send them to the X server.

```

```
    Arguments:
       ITC_id             - in    - Intertask communication identifier
       comm_data          - in    - communication data

    Exceptions:
       none

     High Level PDL
        BEGIN f_cgeh_comm_put
           USE XChangeProperty() TO store comm_data into root window's property
           USE XSync() TO flush output buffer to send request to the X server
           return s_spd_success
        END f_cgeh_comm_put
     END of PDL
========================================================================*/
t_spd_status f_cgeh_comm_put (
                 t_cgeh_comm_ITC_id       *ITC_id,
                 t_cgeh_comm_data         *comm_data)
{
};


/*=======================================================================

    Function:    f_cgeh_comm_is_wanted

    Description:
       This function checks for property change events on the root window
       and determines whether the event (the message) is sent by
       the Generation Server is for him (the Monitor/Control Generation HMI).

    Arguments:
       ITC_id                  - in    - Intertask communication identifier
       event                   - in    - X event

    Exceptions:
       none

     High Level PDL
        BEGIN f_cgeh_comm_is_wanted
           IF event's type == PropertyNotify && event's window == root &&
             event's atom == ITC_id's data atom THEN
               return c_spd_true
           ELSE
               return c_spd_false
           ENDIF
        END f_cgeh_comm_is_wanted
     END of PDL
========================================================================*/
t_spd_boolean f_cgeh_comm_is_want (
                 t_cgeh_comm_ITC_id       *ITC_id,
                 XEvent  *event)
{
};
```

```
/*==========================================================================

  SPD - Signal Processing Demonstrator Computer System

  (C) Copyright Her Majesty the Queen in Right of Canada
  as represented by the Minister of National Defence

  File Name:      cgeh_main.c - Main program of Monitor/Control Generation task.

                  CGEH is a package under the SMC [System Monitor/Control]
                  CSC in the Controller Functional area. CGEH implements
                  the Monitor/Control Geneation task which supports the
                  initiation, termination and monitor the progress of the
                  signal generation activity.

  Identification: [@(#)]

  SCCS Id:        @(#)cgeh_main.c 1.2 93/05/10

  Responsible Organization:

                  MacDonald Dettwiler and Associates

  Purpose:        This file implements the main program for the
                  Monitor/Control Generation task.


  Designer:       Stephen Tran

  Change History:

     SCR  Date         Author   Change made
     ---  -----------  ------   ----------------------------------------
     n/a  29-Jan-1993  cst      Initial Release

===========================================================================*/

/* Included external declarations */
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include "spd_intf.h"
#include "crad_intf.h"
#include "cgeh_intf.h"
#include "cget_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* common constants */
/* none */

/* Implementation */
/*==========================================================================

   program:     cgeh_main

   Description:
      Main program to run Monitor/Control Generation task.

      This is a stand-alone task which provides capability to monitor
      and control signal generation activity through user interface
      on Motif and X window environment.  This task is implemented using
      UIM/X tool with X and Motif widget sets.

      This task is implemented using UIM/X tool with X and Motif widgets.
```

```
UIM/X is a user interface management tool; all code can be entered
without leaving the application itself and UIM/X provides a text
editor function for editing C code. All widget resources (core,
specific, behavior, declaration, constraint) defined for an interface
are kept in an interface file (.i); one file per interface.

These interface files are created automatically by UIM/X. The C code
is created either calling uxcgen utility from the UNIX command line
or pressing the "Write C code" button in UIM/X. Typically, the
interface files are submitted for Configuration Management and
the C code is created for compiling and linking via makefile.

In essence, most of the CGEH CSU codes are implemented in X callback
functions (how the widget reacts to events). The UIM/X tool supports
the generation of the callback function as well as the widget
creation and display function in C code; one .C file per interface.
We have to compile the generated C code and link them into application
executable.

This tasks calls various initialization routines including displaying
the visualization window (top-level window) and then enters
a loop, awaiting x events and never to return.  The user
can terminate this task by selecting the "EXIT" menu.

A signal generation control panel window will be displayed. Before
executing a generation activity, the user must choose a satellite/radar
definition and an environment definition from the database by picking
path and file name from a list of files.  The user will specify
a generation identifier (gen_ident) for the current generation activity.

When the user selects "EXECUTE" command, this program will perform
a validation check on entire satellite/radar definition and environment
defined to ensure there are no errors in the validation of
the parameter data.  This is to prevent the user from performing
a signal generation with errors in the definition parameters
that may generate errorous or unpredictable results in the generation
activity.

The user will be asked for whether to bypass this validation option
such that the validation errors will be ignored and the generation
will be executed.

If no serious errors have been found in the validation check, then
this task will send a generation request message to the Generation
Server Task.  This task is a client to the Generation Server Task.
The communication sending requests to Generation Server utilizes
Remote Procedure Calls (RPC).

Because of resource constraint (the co-processor), only one generation
activity can be activated at any time. The Generation Server is the
guard to accept or reject the generation request.

When the generation request is accepted, this task will monitor
the progress of the generation activity.  The progress status
is sent from the Generation Server through a X property event.
When this task is waken by an X PropertyNotify event, a percent
complete message is read and the progress status will be updated.

All output files corresponding to this generation activity e.g. error
log file, RSS Intermediate Products, Simulated Radar Signal Data (SRSD)
file and RSS derived processing parameters file will have file name
embedded by gen_ident.

While the signal generation is executing, the user can abort the
execution by invoking a "ABORT" command.  The user can either
re-issue a new generation or exit from this task.

Command:  cgeh_main 2>log_pathname &

This task will log unsolicited errors occured in X/Motif calls or
```

```
         RPC calls to standard error and the error messages will then be
         re-directed to the user-specified pathname.  On the other hand,
         the solicited errors, which is associated with a run, like RSS
         generation errors, and validation errors will be output
         to a log file with file name embedded by gen_ident.

      External variables:
         none

      Exceptions:
         Errors logged by RSS and any error occurred in the Controller
         Generation Server for a generation activity will be merged into
         one error file with prefix of gen_ident.

      Notes:
         This file is for the main program only, all dialog creation
         and callback functions will be included in different files
         (generated by UIM/X) in this package; one file per interface.

      High Level PDL
         BEGIN cgeh_main
            get node name of Controller and RSS from environment variables:
               CTLR_NODE and RSS_NODE
            get default directories for input files and output products
               from environment variables
            get node name, user name and pid

            set defaults to input files, output directory and gen_ident

            USE XtAppInitialize TO initialize the X Application
            USE f_cgeh_comm_initialize() TO initialize dialogue communications

            create shell widget and display top-level window

            USE clnt_create TO create RPC connection to Controller
                 Generation Server task
            IF the RPC program/version number is not registered THEN
               display a confirmation window and notify user that the server
                 is not running
            ENDIF

            USE XSelectInput() TO request PropertyChangeMask event for root window
            IF initialization was not successful THEN
                display a confirmation window and notify user errors
            ENDIF

            FOR (infinite loop)
                USE XtAppNextEvent TO get next X event
                USE f_cgeh_comm_is_wanted() TO check if event is wanted
                IF is wanted THEN
                    USE f_cgeh_comm_get() TO retrieve communication data structure
                        from root window's property
                    CASE message type
                        c_cgeh_gen_compl,
                        c_cgeh_gen_failed,
                        c_cgeh_abort_compl,
                        c_cgeh_abort_failed:
                              display the text message (start/abort status)
                                to a confirmation window
                         c_cgeh_in_progress :
                              update percent complete widget on the top-level
                                window
                         default:
                    ENDCASE
                ELSE
                    USE XtDispatchEvent() TO dispatch X event
                ENDIF
            ENDFOR

      END cgeh_main

      BEGIN user interface services functions
         { these functions will be implemented in different
           callback functions for separate widgets }

         case user presses "SELECT SAT/RAD" :
            user selects Satellite/Radar Definition pathname (directory +
              filename) through a selection list

         case user presses "SELECT ENV" :
            user selects Environment Definition pathname (directory +
              filename) through a selection list

         case user presses "OUTPUT DIRECTORY" :
            user enters output directory name for RSS Intermediate Products,
               SRSD file, RSS derived processing parameters file.

         case user presses "GEN IDENT" :
            user enters gen_ident to specify the name of the generatin run

         case user presses "EXECUTE" button :
            USE f_cval_sat_rad_def TO validate sat/rad definition and
              environment definition
            IF (error) THEN
               display validation log in the error message area
               display a confirmation window and ask the user whether
                 to proceed or not
               IF not proceed THEN
                  return
               ENDIF
            ENDIF
            send start generation request to Controller Generation Server
              via RPC clnt_call().
            IF no reply after timeout THEN
               display a confirmation window and notify user that the
                 generation server is not running
            ENDIF

         case user presses "ABORT" button :
            send abort generation request to Controller Generation Server
              via RPC clnt_call().
            IF no reply after timeout THEN
               display a confirmation window and notify user that the
                 generation server is not running
            ENDIF

      END user interface service functions
   End of PDL
==========================================================================*/

/* Routine Declaration */
t_spd_status main (
        t_spd_int32         argc,
        char                **argv,
        char                **envp)
{   /* Begin of main */
}   /* End of main */
```

D1.14 Generation Server (CGET)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      cget_intf.x - Controller Generation Server XDR file

                Package CSIT supports Controller Generation Server task
                within the Controller functional area.

Identification: [@(#)]

SCCS Id:        @(#)cget_intf.x 1.2 93/05/10

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file declares the data format of RPC messages sent
                from Controller Monitor/Control Generation HMI task to
                Controller Generation Server.
                This needs rpcgen to produce cget_intf.h and cget_intf_xdr.c files.

Designer:       Stephen Tran

Change History:

    SCR  Date         Author  Change made
    ---  -----------  ------  -----------------------------------------
    n/a  12-Mar-1993   cst    Initial Release

===========================================================================*/

#ifdef RPC_HDR
%#ifndef cget_intf_h
%#define cget_intf_h
%#include "spd_intf.h"
%#include <X11/Intrinsic.h>
#endif

%/*------------------------------------------------------------------------ *
% *                                                                         *
% *    DEFINE CONSTANTS                                                     *
% *                                                                         *
% *------------------------------------------------------------------------*/
const c_cget_max_node_len = 25;
const c_cget_max_usrnm_len = 25;

%/*------------------------------------------------------------------------ *
% *                                                                         *
% *    DEFINE ENUMERATED DATA TYPES                                         *
% *                                                                         *
% *------------------------------------------------------------------------*/
%/*
%   the message type sent from Generation Server to Monitor/Control
%   Generation HMI task.
% */

enum t_cget_gencmd_types {
    c_cget_start_gencmd,          /* start generation */
    c_cget_abort_gencmd,          /* abort generation */
    c_cget_shutdwn_gencmd,        /* shutdown generation */
    c_cget_status_cmd,            /* status request */
    c_cget_ping_cmd               /* ping message */
};


%/*------------------------------------------------------------------------ *
```

```
% *                                                                           *
% *    DEFINE DATA STRUTURES                                                  *
% *                                                                           *
% * ------------------------------------------------------------------------*/
%/*
%    Data structure for start generation request
% */
struct t_cget_gen_parms
{
   /* Client information (Monitor/Control Generation HMI task) */
   char hostname[c_cget_max_node_len];
                                /* host name returned from "hostname" */
   char username[c_cget_max_usrnm_len];
                                /* UNIX user name returned from "whoami" */
   Display     display;         /* Display used by the client */
   t_spd_int32 pid;             /* the client's process ID returned from
                                   getpid */

   /* pathname of input/output files */
   char satdef_pathname[c_spd_path_size];
                                /* Satellite/Radar Definition pathname */
   char envdef_pathname[c_spd_path_size];
                                /* Environment Definition pathname */
   char output_dir[c_spd_path_size];
                                /* Signal Generation Results output directory */
   char gen_ident[c_spd_name_size];
                                /* Signal generation identifier */
};
%/*
%    Data structure for abort generation request
% */
struct t_cget_abort_parms
{
   /* Client information (Monitor/Control Generation HMI task) */
   char hostname[c_cget_max_node_len];
                                /* host name returned from "hostname" */
   char username[c_cget_max_usrnm_len];
                                /* UNIX user name returned from "whoami" */
   t_spd_int32 pid;             /* process ID returned from getpid() */
   char gen_ident[c_spd_name_size];
                                /* Signal generation identifier */
};
%/*
% *
% *   Integers enumerating generation state types
% */
enum t_cget_gen_state
{
   c_cget_running,              /* generation is currently running */
   c_cget_idle                  /* generation is idle */
};
%/*
%    Data structure for status parameters replied from Generation
%    Server with svc_sendreply().
% */
struct t_cget_stat_parms
{
   t_cget_gen_state     state;  /* current generation state */

   /* In the case when the Generation Server is busy, the information
      of current geneation is replied.
      This information is then displayed as a status message in
      the client's dialog window */
   char hostname[c_cget_max_node_len];
                                /* host name */
   char username[c_cget_max_usrnm_len];
                                /* UNIX user name */
   char gen_ident[c_spd_name_size];
                                /* Signal generation identifier */
};
```

```


%/*-----------------------------------------------------------------------
% *
% *   Data structure for Generation command message.
% *
% *   source:   Controller Monitor/Control Generation HMI task, or
% *             startup/shutdown generation task
% *   sink:     Controller Generation Server task
% *
% *   The Generation Command message consists of a command type and command
% *   parameters.  The command type tells which command parameters are
% *   sent from the Controller to the RSS.
% *
% *   The Generation command type can be either:
% *
% *       1) start_gencmd       - start generation parameters follow
% *       2) abort_gencmd       - abort generation parameters follow
% *       3) shutdwn_gencmd     - no parameters follow
% *       4) status_cmd         - no parameters follow
% *       5) ping_cmd           - no parameters follow
% */
union t_cget_gencmd switch (t_cget_gencmd_types command_type)
{
   case c_cget_start_gencmd:            /* start generation message */
      t_cget_gen_parms          gen_parms;

   case c_cget_abort_gencmd:            /* abort generation message */
      t_cget_abort_parms        abort_parms;

   case c_cget_shutdwn_gencmd:          /* shutdown generation message */
      void;

   case c_cget_status_cmd:              /* status request message */
      void;

   case c_cget_ping_cmd:                /* ping message */
      void;
};

#ifdef RPC_HDR
%#endif
#endif
```

```
/*============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      cget_main.c - Main program of Controller Signal Generation
                Server

Identification: [@(#)]

SCCS Id:        @(#)cget_main.c 1.2 93/05/10

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package implements the main program for the
                daemon server for functions in the Controller Signal Generation
                Task (CGET).


Designer:       Stephen Tran

Change History:

   SCR  Date         Author   Change made
   ---  -----------  ------   ----------------------------------------
   ??? 29-Jan-1993  cst      Initial Release

==========================================================================*/

/* Included external declarations */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include <netdb.h>
#include <rpc/rpc.h>
#include <sys/time.h>
#include <sys/socket.h>
#include "spd_intf.h"
#include "crgi_intf.h"
#include "cgeh_intf.h"
#include "cget_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* common constants */

/* Implementation */
/*============================================================================

   program:     cget_main

   Description:
      Main program to run Controller Signal Generation Server Task.
      It is implemented as a daemon (non-terminating process).

      This task is the server of the Controller Monitor/Control
      Generation HMI Task (see package CGEH), while it is the client
      of RSS Generation Manager.  Since there can be many Monitor/Control
      Generation HMI processes launched by multiple users, this task acts
      as a guard to grant or reject simulation run request from different
      Controller Monitor/Control Generation HMI tasks.

      The communications sending generation parameters and abort
      notify messages from CGEH to CGET is through RPC; however,
      the percentage completion message and generation status
      returned to CGEH is through X window property, since the CGEH
      is running under X application.  (The HMI dialogue task will
      get notified by the event type "PropertyNotify").

      In addition, this server also communicates with RSS playback
      Manager Task in the RSS functional area.  The communication
      links and message protocol used between this task and the RSS
      utilizes RPC (Remote Procedure Call) package.

      The messages received and sent by this task is defined in a separate
      package CRGI (Controller/RSS Generation interface).

   Arguments:
      none

   global variables:
      g_svr_state       - generation is currently running or server is idle
      g_clnt_name       - UNIX user name of the person who requested
                          the generation
      g_clnt_host       - client host name
      g_Xdisplay        - X display used by the client
      g_pid             - client process id
      g_ITC_id          - intertask communication identifier

   Exceptions:
      This server will log errors to its own error file, when it can not
      communicate with RSS.

      At the end of the generation (success or failure), this server
      appends the error file generated by RSS during the signal generation
      activity to its own error log file.
      The generation identifier (gen_ident) will be embedded in the
      file name of error log.

   Notes:
      <algorithm, warning, assumption, etc.>

   High Level PDL
      BEGIN cget_main
        global g_svr_state = c_cget_idle
        USE pmap_unset TO un-register previous incarnations of RPC program
          number and RPC version number
        USE svc_register TO register the transport services
          for Controller port and RSS port
        USE svctcp_create TO create TCP receiving ports for
          Controller and RSS
        USE clnt_create TO create a client object for communications
          to RSS Generation Manager
        USE svc_run TO infinite loop for serving incoming messages
          either from Controller or RSS
      END cget_main


      BEGIN cget_dispatch_ctlr
        USE svc_getargs TO read the incoming message from Controller tasks
          (Generation HMI or start/shutdown program)

        CASE command type DO
          c_cget_start_gencmd:
            USE svc_sendreply() TO reply xdr_void to the client
            IF global g_svr_state == c_cget_idle THEN
              g_clnt_host = message's hostname
              g_clnt_name = message's username
              g_clnt_pid  = message's pid
              g_Xdisplay = message's Xdisplay
              g_gen_ident = message's gen_ident
              USE f_cerr_initialize to initialize error log file
        USE f_cgeh_comm_initialize TO register X atoms using
          g_Xdisplay, g_clnt_pid
        USE clnt_call TO issue "start generation" command to RSS
        USE f_cgeh_comm_put TO send "generation started" to CGEH task
        USE f_cgeh_comm_put TO send percent=0% to CGEH task
        g_svr_state = c_cget_running
      ELSE
        USE f_cgeh_comm_put TO tell Controller Signal Generation HMI
          that RSS is busy
      ENDIF

    c_cget_abort_gencmd:
      USE svc_sendreply() TO reply xdr_void to the client
      IF (global g_svr_state != c_cget_running) or
         (not the same user or not the same gen_ident) THEN
        { the user can only abort his own }
        USE f_cgeh_comm_put TO tell "can't abort" by sending
          message with msg_type=c_cgeh_abort_failed to CGEH task
      ELSE
        USE clnt_call TO send "abort generation" command to RSS
      ENDIF

    c_cget_shutdwn_cmd:
      IF g_svr_state == c_cget_running THEN
        USE clnt_call TO send "abort generation" command to RSS
      ENDIF
      USE f_cerr_log TO log "shutdown by request" to generation
        error log file
      USE f_cerr_merge_logs TO merge RSS and generation error log files
      USE svc_sendreply() TO reply xdr_void to the client
      USE exit() TO terminate the Controller Simulation Server

    c_cget_status_cmd:
      USE svc_sendreply() TO reply current status (g_svr_state)
        to the sender

    c_cget_ping_msg:
      USE svc_sendreply() TO reply xdr_void to the sender

    NOTE: any error returned from f_cgeh_comm_put will be treated
          as as normal; i.e. the HMI client quits and the
          process is not there any longer
END cget_dispatch_ctlr

BEGIN cget_dispatch_RSS
  USE svc_getargs TO read the incoming response message from RSS

  CASE response type DO

    c_crgi_start_genrsp:
      IF status  = c_crgi_in_progress THEN
        USE f_cgeh_comm_put TO send message with
          msg_type=c_cgeh_in_progress and percent received from
          RSS to CGEH task
      ELSE IF status == c_crgi_gen_compl THEN
        USE f_cerr_merge_logs TO merge RSS and generation error log files
        USE f_cgeh_comm_put TO send message with
          msg_type=c_cgeh_gen_compl and percent=100% to CGEH task
        g_svr_state = c_cget_idle
      ELSE IF status == c_crgi_gen_failed THEN
        USE f_cerr_merge_logs TO merge RSS and generation error log files
        USE f_cgeh_comm_put TO send message with
          msg_type=c_cgeh_gen_failed and percent to CGEH task
        g_svr_state = c_cget_idle
      ENDIF

    c_crgi_abort_genrsp:
      IF status == c_crgi_abort_compl
        USE f_cerr_merge_logs TO merge RSS and generation error log files
        USE f_cgeh_comm_put TO send message with
```

```
                msg_type=abort_compl and percent received from RSS to CGEH task
              g_svr_state = c_cget_idle
            ELSE IF status == c_crgi_abort_failed
              USE f_cerr_merge_logs TO merge RSS and generation error log files
              USE f_cgeh_comm_put TO send message with
                msg_type=c_cgeh_abort_failed and percent received from
                RSS to CGEH task
            END IF

          NOTE: any error returned from f_cgeh_comm_put will be treated
                as as normal; i.e. the HMI client quits and the
                process is not there any longer
      END cget_dispatch_RSS
   End of PDL
==========================================================================*/

/* Routine Declaration */
t_spd_status main (
      t_spd_int32       argc,
      char              **argv,
      char              **envp)
{   /* Begin of main */
}   /* End of main */
```

THIS PAGE INTENTIONALLY LEFT BLANK

D1.15 Control Simulation HMI (CSIH)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csih_comm.h - Controller Simulation HMI External Include file
                              for X window communications

Identification: [@(#)]

SCCS Id:        @(#)csih_intf.h 1.1 93/04/22

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file declares funtion description for communicating
                with the Controller Simulation HMI X window.

Designer:       Stephen Tran

Change History:

    SCR  Date         Author   Change made
    ---  -----------  ------   ------------------------------
    n/a  16-Feb-1993  cst      Initial Release

=========================================================================*/

#ifndef csih_intf_h
#define csih_intf_h
#include <X11/Xlib.h>

/*------------------------------------------------------------------ *
 *                                                                   *
 *    DEFINE CONSTANTS                                               *
 *                                                                   *
 *------------------------------------------------------------------*/
#define c_csih_comm_data_name    "csih_data_"
#define c_csih_comm_type_name    "csih_type_"


/*------------------------------------------------------------------ *
 *                                                                   *
 *    DEFINE ENUMERATED DATA TYPES                                   *
 *                                                                   *
 *------------------------------------------------------------------*/
/*
   the message type sent from Simulation Server to Monitor/Control
   Simulation HMI task.
 */

typedef enum {
    c_csih_sim_compl,           /* simulation completed message */
    c_csih_sim_failed,          /* simulation failed message */
    c_csih_abort_compl,         /* abort completed message */
    c_csih_abort_failed,        /* abort failed message */
    c_csih_in_progress          /* simulation progress message */
} t_csih_comm_msg_types;


/*------------------------------------------------------------------ *
 *                                                                   *
 *   DEFINE DATA STRUTURES                                           *
 *                                                                   *
 * ------------------------------------------------------------------*/
/*
    Data structure to be stored in a property.
```

D-241

```
    Used in Dialogue intertask communication
        msg_type          tag indicating what event has occurred
        iteration_no      the iteration number just completed
        elapsed_time      the elapsed time upto the end of last iteration
 */
typedef struct {
    t_csih_comm_msg_types       msg_type;
    t_spd_int32                 iteration_no;
    t_spd_real                  elapsed_time;
} t_csih_comm_data;

/*
   Intertask communication identifier
 */
typedef struct {
    Display *display;       /* connection to the X server              */
    Atom    data_atom;      /* atom representing property name         */
    Atom    data_type_atom; /* atom representing data type in property */
} t_csih_comm_ITC_id;

/*------------------------------------------------------------------------- *
 *                                                                          *
 *    DEFINE FUNCTION SPECIFICATION                                         *
 *                                                                          *
 *--------------------------------------------------------------------------*/

/*========================================================================

   Function:    f_csih_comm_initialize

   Description:
      This function creates X atoms (data atom and data type atom)
      using appended pid (process id of the monitor/Control Simulation
      HMI task) and returns an intertask communication identifier (ITC_id).
      The returned ITC_id will be used in the subsquent calls in
      f_csih_comm_put and f_csih_comm_get.

   Arguments:
      display                    - in    - X Display structure; returned from
                                           XOpenDisplay
      pid                        - in    - pid of Monitor/Control Simulation
                                           HMI task
      ITC_id                     -    out - intertask communication identifier

   Exceptions:
      s_spd_failure returned if it failed to create atoms

   Note:
      This must be called by both Monitor/Control Simulation HMI task
      and Controller Generataion server.  The pid of the Monitor/Control
      Simulation must be known by the Controller Simulation Server
      in order to establish the x communication.

 ========================================================================*/
extern t_spd_status f_csih_comm_initialize (
                Display *display,
                t_spd_int32 pid,
                t_csih_comm_ITC_id *ITC_id);

/*========================================================================

   Function:    f_csih_comm_get

   Description:
      This function retrieves communication data from root window's property.

   Arguments:
      ITC_id          - in    - Intertask communication identifier
      comm_data       -    out - communication data

    Exceptions:
       s_spd_failure returned if it failed to read root window property.

 ==========================================================================*/
extern t_spd_status-f_csih_comm_get (
                t_csih_comm_ITC_id        *ITC_id,
                t_csih_comm_data          *comm_data);

/*==========================================================================

    Function:     f_csih_comm_put

    Description:
       This function stores communication data into root window's property
       and then send them to the X server.

    Arguments:
       ITC_id             - in     - Intertask communication identifier
       comm_data          - in     - communication data

    Exceptions:
       none

 ==========================================================================*/
extern t_spd_status f_csih_comm_put (
                t_csih_comm_ITC_id        *ITC_id,
                t_csih_comm_data          *comm_data);


/*==========================================================================

    Function:     f_csih_comm_is_wanted

    Description:
       This function checks for property change events on the root window
       and determines whether the event (the message) is sent by
       the Simulation Server is for him (the Monitor/Control Simulation HMI).

    Arguments:
       ITC_id                      - in     - Intertask communication identifier
       event                       - in     - X event

    Exceptions:
       none

 ==========================================================================*/
extern t_spd_boolean f_csih_comm_is_want (
                t_csih_comm_ITC_id        *ITC_id,
                XEvent  *event);

#endif  /* end of csih_intf_h */
```

```
/*===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csih_comm.c - Intertask communication from
                              Controller Simulation server task to
                              Controller Monitor/Control Simulation HMI task.

Identification: [@(#)]

SCCS Id:        @(#)csih_comm.c 1.1 93/04/22

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file implements the intertask communication service
                functions using X property to facilitate communication
                from Controller Simulation Server task to Controller
                Monitor/Control Simulation HMI task which is waiting for
                X events.

Designer:       Stephen Tran

Change History:

   SCR  Date         Author   Change made
   ---  -----------  ------   ------------------------------
   ??? 09-Mar-1993  cst      Initial Release

===========================================================================*/

/* Included external declarations */
#include <X11/Xlib.h>
#include "spd_intf.h"
#include "csih_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* common constants */
/* none */

/* Implementation */
/*===========================================================================

   Function:    f_csih_comm_initialize

   Description:
      This function creates X atoms (data atom and data type atom)
      using appended pid (process id of the monitor/Control Simulation
      HMI task) and returns an intertask communication identifier (ITC_id).
      The returned ITC_id will be used in the subsquent calls in
      f_csih_comm_put and f_csih_comm_get.

   Arguments:
      display                  - in    - X Display structure; returned from
                                         XOpenDisplay
      pid                      - in    - pid of Monitor/Control Simulation
                                         HMI task
      ITC_id                   -   out - intertask communication identifier

   Exceptions:
      s_spd_failure returned if it failed to create atoms

```

```
     Note:
        This must be called by both Monitor/Control Simulation HMI task
        and Controller Generataion server.  The pid of the Monitor/Control
        Simulation must be known by the Controller Simulation Server
        in order to establish the x communication.

      High Level PDL
         BEGIN f_csih_comm_initialize
            USE XInternAtom() TO create data atom with name
               (c_csih_comm_data_name + pid string)
            USE XInternAtom() TO create data type atom with name
               (c_csih_comm_type_name + pid string)
            IF atom creation is successful THEN
               store display, data atom, data type atom to ITC_id
               return s_spd_success
            ELSE
               return s_spd_failure
            ENDIF
         END f_csih_comm_initialize
      END of PDL
  ========================================================================*/
t_spd_status f_csih_comm_initialize (
                Display *display,
                t_spd_int32 pid,
                t_csih_comm_ITC_id *ITC_id)
{
};

/*========================================================================

    Function:    f_csih_comm_get

    Description:
       This function retrieves communication data from root window's property.

    Arguments:
       ITC_id            - in    - Intertask communication identifier
       comm_data         -    out - communication data

    Exceptions:
       s_spd_failure returned if it failed to read root window property.

     High Level PDL
        BEGIN f_csih_comm_get
           USE XGetWindowProperty() TO retrieve comm data from root window's
               property
           IF retrieval is successful THEN
               copy comm data to comm_data
               USE XFree() TO free data storage
               return s_spd_success
           else
               return s_spd_failure
           end if
        END f_csih_comm_get
     END of PDL
 ========================================================================*/
t_spd_status f_csih_comm_get (
                t_csih_comm_ITC_id      *ITC_id,
                t_csih_comm_data        *comm_data)
{
};

/*========================================================================

    Function:    f_csih_comm_put

    Description:
       This function stores communication data into root window's property
       and then send them to the X server.

```

```
   Arguments:
      ITC_id                - in    - Intertask communication identifier
      comm_data             - in    - communication data

   Exceptions:
      none

    High Level PDL
       BEGIN f_csih_comm_put
          USE XChangeProperty() TO store comm_data into root window's property
          USE XSync() TO flush output buffer to send request to the X server
          return s_spd_success
       END f_csih_comm_put
    END of PDL
====================================================================*/
t_spd_status f_csih_comm_put (
               t_csih_comm_ITC_id      *ITC_id,
               t_csih_comm_data        *comm_data)
{
};


/*=====================================================================

   Function:    f_csih_comm_is_wanted

   Description:
      This function checks for property change events on the root window
      and determines whether the event (the message) is sent by
      the Simulation Server is for him (the Monitor/Control Simulation HMI).

   Arguments:
      ITC_id                       - in    - Intertask communication identifier
      event                        - in    - X event

   Exceptions:
      none

    High Level PDL
       BEGIN f_csih_comm_is_wanted
          IF event's type == PropertyNotify && event's window == root &&
            event's atom == ITC_id's data atom THEN
              return c_spd_true
          ELSE
              return c_spd_false
          ENDIF
       END f_csih_comm_is_wanted
    END of PDL
====================================================================*/
t_spd_boolean f_csih_comm_is_want (
               t_csih_comm_ITC_id      *ITC_id,
               XEvent  *event)
{
};
```

```
/*===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csih_main.c - Main program of Controller
                              Simulation HMI task.

Identification: [@(#)]

SCCS Id:        @(#)csih_main.c 1.1 93/04/22

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file implements the main program for the
                Simulation HMI task.


Designer:       Stephen Tran

Change History:

    SCR  Date         Author  Change made
    ---  -----------  ------  ------------------------------------------
    ???  29-Jan-1993  cst     Initial Release

============================================================================*/

/* Included external declarations */
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>
#include <stdio.h>
#include "spd_intf.h"
#include "csid_intf.h"
#include "csih_intf.h"
#include "csit_intf.h"
/* #include "cgep.h" */
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* common constants */
/* none */

/* Implementation */
/*=========================================================================

   program:     csih_main

   Description:
      Main program to run Controller Simulation HMI task.

      This is a stand-alone task which provides capability to monitor
      and control simulation run activity through user interface
      on Motif and X window environment.  This task is implemented using
      UIMX tool with X and Motif.

      This task is implemented using UIM/X tool with X and Motif widgets.
      UIM/X is a user interface management tool; all code can be entered
      without leaving the application itself and UIM/X provides a text
      editor function for editing C code. All widget resources (core,
        specific, behavior, declaration, constraint) defined for an interface
        are kept in an interface file (.i); one file per interface.

        These interface files are created automatically by UIM/X. The C code
        is created either calling uxcgen utility from the UNIX command line
        or pressing the "Write C code" button in UIM/X. Typically, the
        interface files are submitted for Configuration Management and
        the C code is created for compiling and linking via makefile.

        In essence, most of the CSIH CSU codes are implemented in X callback
        functions (how the widget reacts to events). The UIM/X tool supports
        the simulation of the callback function as well as the widget
        creation and display function in C code; one .C file per interface.
        We have to compile the generated C code and link them into application
        executable.

        This tasks calls various initialization routines and then enters
        XtAppMainLoop, awaiting x events and never to return.  The user
        can terminate this task by selecting the "EXIT" menu.

        A simulation run control panel window will be displayed. Before
        executing a simulation activity, the user must choose a simulation
        run definition and a Simulated Radar Signal Data (SRSD) file
        from the database by picking path and file name from a list of
        available files.  The user will specify a simulation identifier
        (run_ident) for the current simulation activity.

        When the user selects "EXECUTE" command, this program will perform
        a validation check on entire Simulation Run Definition as well as
        consistency between the Simulation Run Definition and the RSS
        derived processing parameters file header (subset of the
        Satellite/Radar Definition) to ensure there are no errors
        in the validation of the parameter data.  This is to prevent the
        user from performing a simulation run with errors in the
        definition parameters that may generate errorous or unpredictable
        results in the simulation activity.

        The user will be asked for whether to bypass this validation option
        such that the validation errors will be ignored and the simulation
        run will be executed.

        If no serious errors have been found in the validation check, then
        this task will send a simulation request message to the Simulation
        Server Task.  This task is a client to the Simulation Server Task.
        The communication sending requests to Simulation Server utilizes
        Remote Procedure Calls (RPC).

        Because of resource constraint (the RTSG and DSP), only one simulation
        run activity can be activated at any time. The Simulation Server is the
        guard to accept or reject the simulation request.

        When the simulation request is accepted, this task will monitor
        the progress of the simulation activity.  The progress status
        is sent from the Simulation Server through a X property event.
        When this task is waken by an X PropertyNotify event, a status
        message including the last iteration number and elapsed time
        is read and the progress status will be updated.

        All output files corresponding to this simulation activity,
        e.g. status file, DSP Intermediate Products, and detections data
        file will have a file name embedded by run_ident.

        While the simulation run is executing, the user can abort the
        execution by invoking a "ABORT" command.  The user can either
        re-issue a new simulation run or exit from this task.

        Command:  csih_main 2>log_pathname &

        This task will log unsolicited errors occured in X/Motif calls or
        RPC calls to standard error and the error messages will then be
        re-directed to the user-specified pathname.  On the other hand,
        the solicited errors, which is associated with a run, like RSS
        playback errors, DSP errors, and validation errors will be output
        to a log file with file name specified by run_ident.

     Global variables:
        g_sim_def          - in-memory Simulation Run Definition data structure
        g_gen_header       - in-memory Generation Header (from RSS derived
                             processing parameters file) which contains a subset
                             of Satellite/Radar Definition
        g_DSP_parms        - in-memory DSP parameters data structure

     Exceptions:
        Errors logged by RSS and any error occurred in the Controller
        Simulation Server for a simulation activity will be merged into
        one error file with prefix of run_ident.

     Notes:
        This file is for the main program only, all dialog creation
        and callback functions will be included in different files
        (generated by UIM/X) in this package; one file per interface.

     High Level PDL
        BEGIN csih_main
           get node name of Controller and RSS from environment variables:
              CTLR_NODE and RSS_NODE
           get default directories for input files and output products
              from environment variables
           get node name, user name and pid

           set defaults to input files, output directory and run_ident

           USE XtAppInitialize TO initialize the X Application
           USE f_csih_comm_initialize() TO initialize dialogue communications

           create shell widget and display top-level window

           USE clnt_create TO create RPC connection to Controller
                Simulation Server task
           IF the RPC program/version number is not registered THEN
              display a confirmation window and notify user that the server
                is not running
           ENDIF

           USE XSelectInput() TO request PropertyChangeMask event for root window
           IF initialization was not successful THEN
               display a confirmation window and notify user errors
           ENDIF

           FOR (infinite loop)
               USE XtAppNextEvent TO get next X event
               USE f_csih_comm_is_wanted() TO check if event is wanted
               IF is wanted THEN
                   USE f_csih_comm_get() TO retrieve communication data structure
                       from root window's property
                   CASE message type
                      c_csih_sim_compl,
                      c_csih_sim_failed,
                      c_csih_abort_compl,
                      c_csih_abort_failed:
                             display the text message (start/abort status)
                               to a confirmation window
                       c_csih_in_progress :
                             update progress status on the top-level window
                             using the iteration number and elapsed time
                      default:
                   ENDCASE
               ELSE
                   USE XtDispatchEvent() TO dispatch X event
               ENDIF
```

```
         ENDFOR

      END csih_main

      BEGIN user interface services functions
         { these functions will be implemented in different
           callback functions for separate widgets }

         case user presses "SELECT SRSD" :
            user selects SRSD pathname (directory + filename)
              through a selection list

         case user presses "SELECT RSS Proc" :
            user selects RSS derived processing parameters pathname
            (directory + filename) through a selection list
            read header from RSS derived processing parameters into g_gen_header

         case user presses "SELECT SIM Def" :
            user selects Simulation Run Definition pathname
            (directory + filename) through a selection list
            USE f_csid_read TO read the specified data file into
              in-memory definition data structure

         case user presses "OUTPUT DIRECTORY" :
            user enters output directory name for ESP/RSP Intermediate Products,
               DSP parameters file

         case user presses "RUN IDENT" :
            user enters run_ident to specify the name of the simulation run

         case user presses "EXECUTE" button :
            USE f_cval_sim_def TO validate Simulation Run Definition and
            consistency against g_gen_header
            IF (error) THEN
               display validation log in the error message area
               display a confirmation window and ask the user whether
                 to proceed or not
               IF not proceed THEN
                  return
               ENDIF
            ENDIF
            send start simulation request to Controller Simulation Server
              via RPC clnt_call().
            IF no reply after timeout THEN
               display a confirmation window and notify user that the
                 simulation server is not running
            ENDIF

         case user presses "ABORT" button :
            send abort simulation request to Controller Simulation Server
              via RPC clnt_call().
            IF no reply after timeout THEN
               display a confirmation window and notify user that the
                 simulation server is not running
            ENDIF

      END user interface service functions
   End of PDL
===========================================================================*/

/* Routine Declaration */
t_spd_status main (
      t_spd_int32       argc,
      char              **argv,
      char              **envp)
{   /* Begin of main */
}   /* End of main */
```

D1.16 Simulation Server (CSIT)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csit_intf.x - Controller Simulation Server XDR file

                Package CSIT supports Controller Simulation Server task
                within the Controller functional area.

Identification: [@(#)]

SCCS Id:        @(#)csit_intf.x 1.1 93/04/22

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file declares the data format of RPC messages sent
                from Controller Monitor/Control Simulation HMI task to
                Controller Simulation Server.
                This needs rpcgen to produce csit_intf.h and csit_intf_xdr.c files.

Designer:       Stephen Tran

Change History:

    SCR   Date          Author   Change made
    ---   -----------   ------   ----------------------------------------
    n/a   12-Mar-1993   cst      Initial Release

============================================================================*/

#ifdef RPC_HDR
%#ifndef csit_intf_h
%#define csit_intf_h
%#include "spd_intf.h"
#endif

%/*-------------------------------------------------------------------------- *
% *                                                                           *
% *    DEFINE CONSTANTS                                                       *
% *                                                                           *
% *--------------------------------------------------------------------------*/
const c_csit_max_node_len = 25;
const c_csit_max_usrnm_len = 25;

%/*-------------------------------------------------------------------------- *
% *                                                                           *
% *    DEFINE ENUMERATED DATA TYPES                                           *
% *                                                                           *
% *--------------------------------------------------------------------------*/
%/*
%   the message type sent from Simulation Server to Monitor/Control
%   Simulation HMI task.
% */

enum t_csit_simcmd_types {
    c_csit_start_simcmd,         /* start simulation */
    c_csit_abort_simcmd,         /* abort simulation */
    c_csit_shutdwn_simcmd,       /* abort simulation */
    c_csit_status_cmd,           /* status request */
    c_csit_ping_cmd              /* ping message */
};


%/*-------------------------------------------------------------------------- *
% *                                                                           *
% *    DEFINE DATA STRUTURES                                              *
% *                                                                       *
% * ----------------------------------------------------------------------*/
%/*
%    Data structure for start simulation request
% */
struct t_csit_sim_parms
{
   /* Client information (Monitor/Control Simulation HMI task) */
   char hostname[c_csit_max_node_len];
                              /* host name returned from "hostname" */
   char username[c_csit_max_usrnm_len];
                              /* UNIX user name returned from "whoami" */
   t_spd_int32 pid;           /* process ID returned from getpid() */

   /* pathname of input/output files */
   char simdef_pathname[c_spd_path_size];
                              /* Simulation Run Definition pathname */
   char SRSD_pathname[c_spd_path_size];
                              /* Simulated Radar Signal Data pathname */
   char RSSproc_pathname[c_spd_path_size];
                              /* RSS derived processing parameters pathname */
   char output_dir[c_spd_path_size];
                              /* Signal Simulation Results output directory */
   char run_ident[c_spd_name_size];
                              /* Signal simulation run identifier */
};
%/*
%    Data structure for abort simulation request
% */
struct t_csit_abort_parms
{
   /* Client information (Monitor/Control Simulation HMI task) */
   char hostname[c_csit_max_node_len];
                              /* host name returned from "hostname" */
   char username[c_csit_max_usrnm_len];
                              /* UNIX user name returned from "whoami" */
   t_spd_int32 pid;           /* process ID returned from getpid() */
   char run_ident[c_spd_name_size];
                              /* Signal simulation run identifier */
};
%/*
% *
% *   Integers enumerating simulation state types
% */
enum t_csit_sim_state
{
   c_csit_running,            /* simulation is currently running */
   c_csit_idle                /* simulation is idle */
};
%/*
%    Data structure for status parameters replied from Simulation
%    Server with svc_sendreply().
% */
struct t_csit_stat_parms
{
   t_csit_sim_state    state; /* current simulation state */

   /* Client (Monitor/Control Simulation HMI task) information */
   char hostname[c_csit_max_node_len];
                              /* host name returned from "hostname" */
   char username[c_csit_max_usrnm_len];
                              /* UNIX user name returned from "whoami" */
   char run_ident[c_spd_name_size];
                              /* Signal simulation run identifier */
};


%/*-------------------------------------------------------------------
% *
% *   Data structure for Simulation command message.
% *
% *   source:    Controller Monitor/Control Simulation HMI task, or
% *              startup/shutdown simulation task
% *   sink:      Controller Simulation Server task
% *
% *   The Simulation Command message consists of a command type and command
% *   parameters.  The command type tells which command parameters are
% *   sent from the Controller to the RSS.
% *
% *   The Simulation command type can be either:
% *
% *       1) start_simcmd       - start simulation parameters follow
% *       2) abort_simcmd       - abort simulation parameters follow
% *       3) shutdwn_simcmd     - no parameters follow
% *       4) status_cmd         - no parameters follow
% *       5) ping_cmd           - no parameters follow
% */
union t_csit_simcmd switch (t_csit_simcmd_types command_type)
{
   case c_csit_start_simcmd:              /* start simulation message */
      t_csit_sim_parms          sim_parms;

   case c_csit_abort_simcmd:              /* abort simulation message */
      t_csit_abort_parms        abort_parms;

   case c_csit_shutdwn_simcmd:            /* shutdown simulation message */
      void;

   case c_csit_status_cmd:                /* status request message */
      void;

   case c_csit_ping_cmd:                  /* ping message */
      void;
};

#ifdef RPC_HDR
%#endif
#endif
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csit_main.c - Main program of Controller Simulation Server

Identification: [@(#)]

SCCS Id:        @(#)csit_main.c 1.2 93/06/16

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package implements the main program for the
                daemon server for functions in the Controller Simulation
                Task (CSIT).


Designer:       Stephen Tran

Change History:

   SCR  Date        Author  Change made
   ---  ----------- ------  ----------------------------------------
   ??? 29-Jan-1993  cst    Initial Release

==========================================================================*/

/* Included external declarations */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include <netdb.h>
#include <rpc/rpc.h>
#include <sys/time.h>
#include <sys/socket.h>
#include "spd_intf.h"
#include "crpi_intf.h"
#include "cdsi_intf.h"
#include "csit_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* common constants */

/* Implementation */
/*==========================================================================

   program:     csit_main

   Description:
      Main program to run Controller Simulation Server Task

      This task is a daemon server spawned by the Controller Monitor/Control
      Simulation HMI Task (see package CSIH), while it is the client of
      RSS Playback Manager.  Since there can be many Monitor/Control
      Simulation HMI processes launched by multiple users, this task acts
      as a guard to grant or reject simulation run request from different
      Controller Monitor/Control Simulation HMI tasks.

      The communications sending simulation parameters and abort
      notify messages from CSIH to CSIT is through RPC; however,
    the percentage completion message returned to CSIH is through
    X window property since the CSIH is running under X application.
    (The HMI dialogue task will get notified by the event type
    "PropertyNotify").

    In addition, this server also communicates with RSS playback
    Task in the RSS functional area and ESP and DSP Executive
    Controller in the DSP functional area.  The communication
    links and message protocol used between this task and the RSS
    and DSP tasks utilizes RPC (Remote Procedure Call).

    The messages received and sent by this task is defined in separate
    packages CRPI (interface to RSS) and CDSI (interface to DSP).

 Arguments:
    none

 Global variables:
    g_svr_state       - current state of the simulation server
                        IDLE     currently, there is no simulation activity
                                 running; i.e. the server can accept the
                                 simulation request from the Monitor/Control
                                 Simulation HMI task.
                        RUNNING there is a simulation currently running.
                                 The server will deny the simulation request
                                 from the simulation HMI task.
    g_clnt_name       - UNIX user name of the person who requested
                        the generation
    g_clnt_host       - client host name
    g_pid             - client process id
    g_ITC_id          - intertask communication identifier
    g_SRSD_pathname   - pathanme of SRSD file of the current run or
                        the last run
    g_SRSD_mtime      - the last modify time of the SRSD file
    g_simdef_pathname - pathname of the simulation run definition file
    g_simdef_mtime    - the last modify time of the simulation run
                        definition file of the current run or the last run
    g_RSSproc_pathname- pathname of RSS generated processing parameters file
                        of the current run or the last run
    g_RSSproc_mtime   - the last modify time of the RSS generated processing
                        parameters file of the current run or the last run



 Exceptions:
    This server will log communication errors when it can not
    communicate with either RSS or DSP (up or down stream),
    or can not send status to the simulation HMI task.

    This server also merges the error log files generated by both RSS
    and DSP during the simulation/playback activity to its own error
    log file.  The run identifier (RUN_IDENT) will be used to generate
    the file name for the error log file under the user's directory
    (passing via the simulation request).

 Notes:
    The server is spawned by the simulation run HMI and running under
    the SPD System Manager. This task will be shutdown by the simulation
    run HMI.

 High Level PDL
    BEGIN csit_main
      initialize g_svr_state = c_csit_idle
      initialize g_SRSD_pathname, g_simdef_pathname, g_RSSproc_pathname
      initialize g_SRSD_mtime, g_simdef_mtime, g_RSSproc_mtime
      USE pmap_unset TO un-register previous incarnations of RPC program
          number and RPC version number
      USE svc_register TO register the transport services
          for this task
      USE svctcp_create TO create three TCP receiving ports, one each for
      RSS, DSP and the simulation HMI task
  USE clnt_create TO create four client objects, one each for RSS
      comunications, ESP and DSP communications
  USE XOpenDisplay TO connect to X server
  g_Xdisplay = Xdisplay
  USE svc_run TO infinite loop for serving incoming messages
      either from RSS, DSP or the simulation HMI task
END csit_main

BEGIN f_csit_dispatch_ctlr
  { this function will dispath the message sent from the simulation
    HMI task }
  USE svc_getargs TO read the incoming message from simulation HMI task

  CASE command type DO
    c_csit_start_simcmd:
      USE svc_sendreply() TO reply xdr_void to the client
      IF (message's hostname != g_clnt_host) or
      (message's pid != g_clnt_id) THEN
         USE f_cgeh_comm_initialize TO register X atoms using
           g_Xdisplay, g_clnt_pid
         g_ITC_id = returned ITC_id
      ENDIF
      g_clnt_host = message's hostname
      g_clnt_name = message's username
      g_clnt_pid  = message's pid
      g_gen_ident = message's gen_ident
      g_SRSD_pathname = message's SRSD_pathname
      g_RSSproc_pathname = message's RSSproc_pathname
      g_simdef_pathname = message's simdef_pathname

      USE stat TO get file last modify time of SRSD file, RSS derived
      processing parameters file, Simulation Run Definition file

      IF g_svr_state == c_csit_idle THEN
        USE f_cerr_initialize to initialize error log file
        USE clnt_call TO issue provide-status command to ESP
          and read the ESP state reply
        USE clnt_call TO issue provide-status command to RSP
          and read the RSP state reply
        IF (sw_vers_id == blank) OR (sw_vers_id is not the same
            as the one specified by the user) THEN
            store last modify time to g_SRSD_mtime,
              g_simdef_mtime, g_RSSproc_mtime
            read simulation run definition file
            read RSS processing parameter file
            format RPC message for DSP processing parameters
            write DSP processing parameters into file
            USE clnt_call TO send configuration command to ESP
            USE clnt_call TO send configuration command to RSP
        ELSE IF (both ESP and RSP are in READY state) and
            (timestamp of either of SRSD file, RSS derived processing
             parameters file, Simulation Run Definition file
             changes since the last run) THEN
            store last modify time to g_SRSD_mtime,
              g_simdef_mtime, g_RSSproc_mtime
            read simulation run definition file
            read RSS processing parameter file
            format RPC message for DSP processing parameters
            write DSP processing parameters into file
            USE clnt_call TO send update parameters command to ESP
            USE clnt_call TO send update parameters command to RSP
        ELSE IF (both ESP and RSP are in READY state)
            store last modify time to g_SRSD_mtime,
              g_simdef_mtime, g_RSSproc_mtime
            USE clnt_call TO send run command to ESP
            USE clnt_call TO send run command to RSP
        ELSE
            { i.e. either ESP or RSP is not in READY state }
           USE f_cerr_log TO log error
           USE f_cutl_merge_logs TO merge RSS and DSP log files
           USE f_csih_comm_put TO send "simulation failed" message
             to HMI task
        ENDIF

        IF any communications failed THEN
           USE f_cerr_log TO log communication error
           USE f_cutl_merge_logs TO merge RSS and DSP log files
           USE f_csih_comm_put TO send "simulation failed" message
             to HMI task
        ELSE IF replied error_flag == FALSE THEN
           USE f_cutl_merge_logs TO merge RSS and DSP log files
           USE f_csih_comm_put TO send "simulation failed" message
             to HMI task
        ENDIF

      ELSE IF g_svr_state == c_csit_running
        USE f_csih_comm_put TO send "request denied" to HMI task
      ENDIF

    c_csit_abort_simcmd:
      USE svc_sendreply() TO reply xdr_void to the client
      IF (abort message's hostname != g_clnt_host) or
         (abort message's pid != g_clnt_id) THEN
         USE f_cgeh_comm_initialize TO register X atoms using
           g_Xdisplay, g_clnt_pid
         g_ITC_id = returned ITC_id
      ENDIF
      IF g_svr_state != c_csit_running THEN
        USE f_csih_comm_put TO tell simulation HMI task
            that simulation is not running
      ELSE
        IF the simulation run wanted to be aborted is
           the current activity THEN
          USE f_csih_comm_put TO tell simulation HMI task
              that the specified activity is not the current run
        ESLE
          USE clnt_call TO send "abort playback command" to RSS
        ENDIF
      ENDIF

      IF communications failed THEN
         USE f_cerr_log TO communications error
         USE f_cutl_merge_logs TO merge RSS and DSP log files
         USE f_csih_comm_put TO send "abort failed" message to HMI task
      ENDIF

    c_csit_shutdwn_cmd:
      IF g_svr_state == c_csit_running THEN
          USE clnt_call TO send "abort playback command" to RSS
          USE clnt_call TO send "stop command" to ESP
          USE clnt_call TO send "stop command" to RSP
      ENDIF
      IF communications failed THEN
         log errors in communications i.e. RSS/ESP/RSP has died already
      ELSE IF replied error_flag == FALSE THEN
         log errors in ESP or RSP
      ENDIF
      USE f_cutl_merge_logs TO merge RSS and DSP log files
      USE svc_sendreply() TO reply xdr_void to the client
      USE exit() TO terminate the Controller Simulation Server

    c_csit_status_cmd:
      USE svc_sendreply() TO reply current server status to the sender

    c_csit_ping_msg:
      USE svc_sendreply() TO reply xdr_void to the sender

END csit_dispatch_ctlr
```

```
BEGIN csit_dispatch_RSS
  USE svc_getargs TO read the incoming message from RSS

  CASE response type DO
    c_crpi_start_plbrsp:
      IF playback_status == c_crpi_in_progress THEN
        USE f_csih_comm_put TO send "elapsed time" message to HMI task
        the playback elapsed time
      ELSE IF playback_status == c_crpi_sim_compl THEN
        USE clnt_call TO issue stop run request to RSP
        USE clnt_call TO issue stop run request to ESP
        USE f_cutl_merge_logs TO merge RSS and DSP log files
        USE f_csih_comm_put TO send "simulation completed" to HMI task
      ELSE IF playback_status == c_crpi_sim_failed THEN
        USE clnt_call TO issue stop run request to RSP
        USE clnt_call TO issue stop run request to ESP
        USE f_cutl_merge_logs TO merge RSS and DSP log files
        USE f_csih_comm_put TO send "simulation failed" to HMI task
      ENDIF

    c_crpi_abort_plbrsp:
      IF abort_status == c_crpi_abort_compl THEN
        USE clnt_call TO issue stop run request to ESP
        USE clnt_call TO issue stop run request to RSP
        USE f_cutl_merge_logs TO merge RSS and DSP log files
        USE f_csih_comm_put TO tell "abort completed" to HMI task
      ELSE IF abort_status == c_crpi_abort_failed
        USE clnt_call TO issue stop run request to RSP
        USE clnt_call TO issue stop run request to ESP
        USE f_cutl_merge_logs TO merge RSS and DSP log files
        USE f_csih_comm_put TO tell "abort failed" to HMI task
      END IF

END csit_dispatch_RSS

BEGIN csit_dispatch_DSP
  USE svc_getargs TO read the incoming message from DSP

  CASE response type DO
    c_cdsi_DSP_detections: (from RDDH in RSP)
      USE xdr to write detection vectors into the detection file
      IF iteration completed THEN (normal end of iteration)
        close detection file
        re-open detection file
        compute real-time factor rate
        USE f_csih_comm_put TO tell "factor rate" to HMI task
      ENDIF
      IF run completed THEN (normal end of run completion)
        USE clnt_call TO send "stop" command to ESP
        USE clnt_call TO send "stop" command to RSP
        USE f_cutl_merge_logs TO merge RSS and DSP log files
        USE f_csih_comm_put TO tell "run completed" to HMI task
      ENDIF
    c_cdsi_cmpl_ackn: (for Configure and Update command to DSP)
      IF (error_flag == TRUE) THEN
         log error message
      ELSE IF RSP
         check RSP done
      ELSE IF ESP
         check ESP done
      ENDIF
      IF both RSP & ESP done THEN
        USE clnt_call TO send run command to ESP
        USE clnt_call TO send run command to RSP
        USE clnt_call TO send "start playback" command to RSS
      ENDIF
    c_cdsi_DSP_unit_error:
     log error message
     USE clnt_call TO send "abort playback" command to RSS
     IF RSP
```

```
                USE clnt_call TO send "stop" command to ESP
              ELSE IF ESP
                USE clnt_call TO send "stop" command to RSP
              ENDIF
              close detection file
              USE f_cutl_merge_logs TO merge RSS and DSP log files
              USE f_csih_comm_put TO tell "run failed" to HMI task
        END csit_dispatch_RSS
     End of PDL
==========================================================================*/

/* Routine Declaration */
t_spd_status main (
        t_spd_int32       argc,
        char              **argv,
        char              **envp)
{   /* Begin of main */
    printf ("\n\n***** NOT SUPPORTED YET *****\n\n");
    exit(s_spd_success);
}   /* End of main */
```

THIS PAGE INTENTIONALLY LEFT BLANK

D1.17 Data Validation (CVAL)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*==========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      cval_intf.h - Validation Utility External Include file

Identification: [@(#)]

SCCS Id:        @(#)cval_intf.h 1.2 93/09/17

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file declares funtion description for the validation
                service routines in the package CVAL.

Designer:       Stephen Tran

Change History:

    SCR   Date         Author   Change made
    ---   -----------  ------   ---------------------------------------
    n/a   22-Jul-1993  cst      Initial Release

==========================================================================*/

#ifndef cval_intf_h
#define cval_intf_h

/*==========================================================================

    Function:    f_cval_sat_def

    Description:
       This function reads a Satellite/Radar Definition structure and
       performs a thorough validation for the entire definition against
       smart ranges and dependencies, which are specified in the Satellite
       and Radar HMI table in Testbed Preliminary Design Document.

    Arguments:
       sat_def                     - in    - Satellite/Radar Definition structure
       num_msg                     -   out - number of validation messages returned
                                             from this function
       val_log                     -   out - array of validation messages.  The
                                             size of the array is specified by
                                             num_msg.


    External variables:
       none

    Exceptions:
       s_spd_failure is returned, if error was detected in the Satellite/Radar
       Definition structure.

    Notes:
       This function will allocate memory for the validation messages.
       The caller is responsible to free the memory after the validation
       messages have been processed.

       Each validation message is an ASCII string defined as "char" data type and
       terminated by a "null" character (\0).  If a validation message is longer
       than 80 characters, then this message will be splitted into multiple lines
       by adding a carriage-return character "\n" at the end of every 80
       characters.

```

```
   ===========================================================================*/
extern t_spd_status f_cval_sat_def (
#ifdef __STDC__
   t_crad_sat_def *sat_def,
   t_spd_int32    *num_msg,
   char           ***val_log
#endif
);

/*============================================================================

   Function:    f_cval_run_def

   Description:
      This function reads a Simulation Run Definition structure and
      performs a thorough validation for the entire definition against
      smart ranges as well as dependencies on its associated
      Satellite/Radar Definition.  The valid ranges and dependencies are
      specified in the Simulation Run HMI table in Testbed Preliminary
      Design Document.

      The associated Satellite/Radar Definition pathname is stored in
      the RSS Derived Processing Parameters structure; this function
      will read the associated Satellite/Radar Definiton data file into
      memory before the validation is processed.

   Arguments:
      run_def                 - in    - Simulation Run Definition structure
      RSS_process_parms       - in    - RSS Derived Processing Parameters
      num_msg                 -   out - number of validation messages returned
                                        from this function
      val_log                 -   out - array of validation messages.  The
                                        size of the array is specified by
                                        num_msg.

   External variables:
      none

   Exceptions:
      s_spd_failure is returned, if error was detected in the Simulation Run
      Definition structure or the associated Satellite/Radar Definition
      data file can not be read.

   Notes:
      This function will allocate memory for the validation messages.
      The caller is responsible to free the memory after the validation
      messages have been processed.

      Each validation message is an ASCII string defined as "char" data type and
      terminated by a "null" character (\0).  If a validation message is longer
      than 80 characters, then this message will be splitted into multiple lines
      by adding a carriage-return character "\n" at the end of every 80
      characters.

 ==========================================================================*
extern t_spd_status f_cval_run_def (
#ifdef __STDC__
   t_csid_run_def *run_def,
   t_cgep_RSS_process_parms *RSS_process_parms,
   t_spd_int32    *num_msg,
   char           ***val_log
#endif
);

#endif
```

```
□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
□ □ □ □                                       □ □ □ □
□ □ □ □                                       □ □ □ □
□ □ □ □                                       □ □ □ □
  □ □                                           □ □
  □ □              CDADL PROCESSOR              □ □
  □ □                                           □ □
  □ □                                           □ □
  □ □               VERSION 2.4.6               □ □
  □ □                                           □ □
  □ □          SERIAL NUMBER: C38U0011          □ □
  □ □                                           □ □
  □ □             AUTHORIZED USER:              □ □
  □ □             Raytheon - Canada             □ □
  □ □                                           □ □
  □ □                                           □ □
  □ □                                           □ □
  □ □                                           □ □
  □ □                                           □ □
□ □ □ □                                       □ □ □ □
□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
```

D-267

TABLE OF CONTENTS

PAGE NO.

1 FILE cval_run_def.c
4 FILE cval_run_local.c
8 FILE cval_run_config.c
11 FILE cval_run_beam.c
14 FILE cval_run_blanker.c
18 FILE cval_run_norm_equ.c
22 FILE cval_run_pulse_compr.c
24 FILE cval_run_unambig_prf.c
26 FILE cval_run_dpca.c
29 FILE cval_run_non_coh.c
31 FILE cval_run_cfar.c
34 FILE cval_run_data_space.c
40 FILE cval_sat_def.c
42 FILE cval_sat_aux_pos.c
45 FILE cval_sat_beam.c
48 FILE cval_sat_beam_tst.c
50 FILE cval_sat_cfa_grid.c
53 FILE cval_sat_data_capt.c
56 FILE cval_sat_data_space.c
62 FILE cval_sat_decimation.c
64 FILE cval_sat_freq_range.c
67 FILE cval_sat_gen_opt.c
69 FILE cval_sat_overlap.c
72 FILE cval_sat_pc_pos.c
75 FILE cval_sat_pri.c
77 FILE cval_sat_pulse.c
80 FILE cval_sat_rra_param.c
83 FILE cval_sat_ssp.c
85 FILE cval_sat_tap_delays.c
88 FILE cval_log_msg.c
90 REPORT TBD CROSS REFERENCE REPORT
91 REPORT INVOCATION TREE
93 REPORT ERROR SUMMARY

```
/*%%==============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence, 1992

File Name:      cval_run_def.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---  ----------- ------  ------------------------------------------------
n/a  14-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_run_def

PURPOSE:        This function performs the validation of the Simulation Run data.


INPUT                                          SOURCE
-----                                          ------
run_def                                        calling_routine
RSS_process_parms                              calling_routine

OUTPUT                                         DESTINATION
------                                         -----------
num_msg                                        calling_routine
val_log                                        calling_routine

CREATION DATE:  June 14 1993

==============================================================================*/
#include "cval.h"

f_cval_run_def(
   t_csid_run_def           *run_def,
   t_cgep_RSS_process_parms *RSS_process_parms,
   t_spd_int32              *num_msg,
   char                     **val_log
)
{
/*|
   clear the num_msg parameter
```

```
set the return code to c_spd_success
Use f_csid_read to read in the sat_def data structure
IF there was an error reading in the data structure THEN
|   set the return code to c_spd_failure
|   Use f_cval_log_msg with index to message indicating "ERROR reading sat_def file"
|   IF there was an error writing the message THEN
|   |   set the return_code to c_spd_failure
|   ENDIF
ELSEIF run_def->DI.DI_len != sat_def->DI.DI_len THEN
|   Use f_cval_log_msg with index to message indicating\
|   "ERROR incompatible sat_def and run_def files"
|   IF there was an error writing the message THEN
|   |   set the return_code to c_spd_failure
|   ENDIF
ELSE
|   set the error_detected_flag to c_spd_false
|   FOR di = 0 TO run_def->DI.DI_len - 1
|   LOOP
|   |   IF run_def->DI.DI_val[di].NCI.NCI_len != sat_def->DI.DI_val[di].NCI.NCI_len  &&
|   |   no error has been detected THEN
|   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   "ERROR incompatible sat_def and run_def files"
|   |   |   IF there was an error writing the message THEN
|   |   |   |   set the return_code to c_spd_failure
|   |   |   ENDIF
|   |   |   Flag that an error was detected
|   |   ELSE
|   |   |   FOR nci = 0 TO run_def->DI.DI_val[di].NCI.NCI_len - 1
|   |   |   LOOP
|   |   |   |   IF ((run_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_len !=\
|   |   |   |   |   sat_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_len) &&\
|   |   |   |   (error_detected_flag == c_spd_false)) THEN
|   |   |   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   |   |   "ERROR incompatible sat_def and run_def files"
|   |   |   |   |   IF there was an error writing the message THEN
|   |   |   |   |   |   set the return_code to c_spd_failure
|   |   |   |   |   ENDIF
|   |   |   |   |   Flag that an error was detected
|   |   |   |   ENDIF
|   |   |   ENDLOOP
|   |   ENDIF
|   ENDLOOP
|   IF no error was detected THEN
|   |   Use f_cval_run_local(run_def, sat_def) to calculate the variables that\
|   |   are local to the CVAL CSU
|   |   Use f_cval_run_config(run_def, sat_def, num_msg, cur_log_entry) to check the\
|   |   system configuration
|   |   Use f_cval_run_beam(run_def, num_msg, cur_log_entry) to check the main beam\
|   |   configuration
|   |   Use f_cval_run_blanker(run_def, num_msg, cur_log_entry) to check the\
|   |   blanker configuration
|   |   Use f_cval_run_norm_equ(run_def, sat_def, num_msg, cur_log_entry) to check
|   |   the normal equations
|   |   Use f_cval_run_pulse_compr(run_def, num_msg, cur_log_entry) to check the \
|   |   pulse compression
|   |   Use f_cval_run_unambig_prf(run_def, num_msg, cur_log_entry) to check the\
    |  |  unambiguous PRF
    |  |  Use f_cval_run_dpca(run_def, sat_def, num_msg, cur_log_entry) to check the\
    |  |  Doppler and ASAR DPCA
    |  |  Use f_cval_run_non_coh(run_def, num_msg, cur_log_entry) to check \
    |  |  the non-coherent integration
    |  |  Use f_cval_run_cfar(run_def, num_msg, cur_log_entry) to check the CFAR paramaters
    |  |  Use f_cval_run_data_space(run_def, sat_def, num_msg, cur_log_entry) to check\
    |  |  the data space requirements of the run-time def
    |  |  IF an error is returned from any of the above checks THEN
<---------------return a c_spd_failure error code
    |  |  ELSE
<---------------return c_spd_success
    |  |  ENDIF
    |  ENDIF
    |  free the memory used by the sat_def file
    ENDIF
<---return the return_code
*/
}

/* end of cval_run_def.c */
```

```
/*%%=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_run_local.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---  -----------  ------  --------------------------------------------------
n/a  15-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_run_local

PURPOSE:        This function creates the temporary variables required to perform the run-time
consistency check.


 INPUT                                              SOURCE
 -----                                              ------
 run_def                                            calling routine
 sat_def                                            calling routine

 OUTPUT                                             DESTINATION
 ------                                             -----------
 t_cval_run_num_blankers                            CVAL global data
 t_cval_run_num_pulses_lost                         CVAL global data
 t_cval_run_DPCA_mode                               CVAL global data
 t_cval_num_ph_ctr_per_grp                          CVAL global data
 t_cval_num_DPCA_grps                               CVAL global data
 t_cval_run_num_mono                                CVAL global data
 t_cval_run_num_ph_ctrs                             CVAL global data
 t_cval_run_num_channels                            CVAL global data
 t_cval_run_bytes_per_samp                          CVAL global data
 t_cval_run_num_nulled                              CVAL global data
 t_cval_run_num_blanked                             CVAL global data
 t_cval_run_num_dop                                 CVAL global data
 t_cval_run_max_nul_chan                            CVAL global data
 t_cval_run_max_sys_size                            CVAL global data
 t_cval_run_npcu                                    CVAL global data
 t_cval_run_ndoft                                   CVAL global data
```

```
t_cval_run_PRF_min[]                                        CVAL global data
t_cval_run_PRF_max[]                                        CVAL global data
t_cval_run_samps_pri[][]                                    CVAL global data
t_cval_run_PRF[][]                                          CVAL global data
t_cval_run_CPI_time[][]                                     CVAL global data
t_cval_run_num_weight_vectors[][]                           CVAL global data
t_cval_run_nbins[]                                          CVAL global data
t_cval_run_naug_bins[]                                      CVAL global data
t_cval_monopulse_enable                                     CVAL global data

CREATION DATE:  June 15 1993

====================================================================================*/
#include "cval.h"


#define NUM_RG_DOP_BLANKERS run_def->general_config.num_rg_dop_blankers
#define PRF_ENTERED         sat_def->DI.DI_val[di].NCI.NCI_val[nci].PRF_entered
#define NCI_PRF             sat_def->DI.DI_val[di].NCI.NCI_val[nci].PRF
#define RXFE_SAMPLING_RATE  sat_def->antenna_model.main_rxfe.sampling_rate
#define SAMPLES_PER_PRI     sat_def->DI.DI_val[di].NCI.NCI_val[nci].samples_per_PRI
#define NUM_PULSES_PROC     run_def->DI.DI_val[di].num_pulses_proc

f_cval_run_local(
    t_csid_run_def *run_def,
    t_crad_sat_def *sat_def
)
{
/*|
    t_cval_run_num_blankers = max(run_def->ECCM_proc.num_pre_dop_blankers, NUM_RG_DOP_BLANKERS)
    IF sat_def->concept.concept == c_spd_CFA THEN
    |   t_cval_run_DPCA_mode = DPCA_MODE
    |   t_cval_num_ph_ctr_per_grp = NUM_PH_CTR_PER_GRP
    |   t_cval_num_DPCA_grps = NUM_DPCA_GRPS
    |   t_cval_monopulse_enable = c_spd_true
    ELSE
    |   t_cval_run_DPCA_mode = c_spd_DPCA_none
    |   t_cval_run_num_ph_ctr_per_grp = 1
    |   t_cval_run_num_DPCA_grps = 1
    |   t_cval_monopulse_enable = c_spd_false
    ENDIF
    IF t_cval_run_DPCA_mode == c_spd_DPCA_none THEN
    |   t_cval_run_num_pulses_lost = 0
    |   t_cval_run_npcu = 1
    ELSEIF ((t_cval_run_DPCA_mode = c_spd_ASAR_DPCA) ||\
    |   (t_cval_run_num_ph_ctr_per_grp) ||\
    ((t_cval_num_DPCA_grps == 1) && (t_cval_run_monopulse_enable == c_spd_true))) THEN
    |   t_cval_run_num_pulses_lost = sat_def->antenna_model.main_antenna.CFA.PRI_multiplier
    |   t_cval_run_npcu = 2
    ELSE
    |   t_cval_run_num_pulses_lost = sat_def->antenna_model.main_antenna.CFA.PRI_multiplier
    |   t_cval_run_npcu = run_def->ECCM_config.num_MASR_ph_ctr
    ENDIF
    t_cval_run_num_ph_ctrs = t_cval_run_num_DPCA_grps * t_cval_run_num_ph_ctr_per_grp
    t_cval_run_ndoft = 0
    FOR aux = 0 TO sat_def->antenna_model.aux_count - 1
```

```
LOOP
|   FOR tap = 0 TO sat_def->antenna_model.aux_antenna[aux].tap_count - 1
|   LOOP
|   |   t_cval_run_ndoft++
|   ENDLOOP
ENDLOOP
t_cval_run_num_channels = t_cval_run_num_ph_ctrs + t_cval_run_ndoft
t_cval_run_bytes_per_samp = num_channels * (2 * sizeof(t_spd_short))
IF t_cval_monopulse_enable == c_spd_true THEN
|   t_cval_run_num_mono = 2
ELSE
|   t_cval_run_num_mono = 1
ENDIF
IF t_cval_run_DPCA_mode == c_spd_ASAR_DPCA THEN
|   t_cval_run_num_nulled = t_cval_run_num_blankers + t_cval_run_num_mono + 1
|   t_cval_run_num_blanked = NUM_RG_DOP_BLANKERS + 1
|   t_cval_run_num_dop = NUM_RG_DOP_BLANKERS + t_cval_run_num_mono + 1
ELSE
|   t_cval_run_num_nulled = t_cval_run_num_blankers + t_cval_run_num_mono  * t_cval_run_npcu
|   t_cval_run_num_blanked = NUM_RG_DOP_BLANKERS * t_cval_run_npcu
|   t_cval_run_num_dop = NUM_RG_DOP_BLANKERS + t_cval_run_num_mono
ENDIF
t_cval_run_max_nul_chan = 16
t_cval_run_max_sys_size = 40

// calculate min/max PRF
FOR di = 0 TO run_def->DI.DI_len - 1
LOOP
|   t_cval_run_PRF_min[di] = largest positive floating point value
|   t_cval_run_PRF_max[di] = 0
|   FOR nci = 0 TO run_def->DI.DI_val[di].NCI.NCI_len - 1
|   LOOP
|   |   IF PRF_ENTERED == c_spd_true (entered directly) THEN
|   |   |   t_cval_run_samps_pri[di][nci] = round(1.0E03 * RXFE_SAMPLING_RATE / NCI_PRF)
|   |   ELSE
|   |   |   t_cval_run_samps_pri[di][nci] = SAMPLES_PER_PRI
|   |   ENDIF
|   |   t_cval_run_PRF[di][nci] = 1.0E03 * RXFE_SAMPLING_RATE/t_cval_run_samps_pri[di][nci]
|   |   IF (t_cval_run_PRF[di][nci] < t_cval_run_PRF_min[di]) THEN
|   |   |   t_cval_run_PRF_min[di] = t_cval_run_PRF[di][nci]
|   |   ENDIF
|   |   IF (t_cval_run_PRF[di][nci] > t_cval_run_PRF_max[di]) THEN
|   |   |   t_cval_run_PRF_max[di] = t_cval_run_PRF[di][nci]
|   |   ENDIF
|   |   t_cval_run_CPI_time[di][nci] = NUM_PULSES_PROC / t_cval_run_PRF[di][nci]
|   |   t_cval_run_num_weight_vectors[di][nci] = floor(t_cval_run_CPI_time[di][nci] *\
|   |   run_def->ECCM_proc.num_update_ints / run_def->ECCM_proc.snapshot_smooth_period) \
|   |   - run_def->ECCM_proc.num_update_ints + 1
|   ENDLOOP
|   IF (run_def->radar_proc.PD_enable == c_spd_true) THEN
|   |   t_cval_run_nbins[di] = run_def->DI.DI_val[di].PD.PD_FFT_len
|   |   t_cval_run_naug_bins[di] = run_def->DI.DI_val[di].PD.PD_FFT_len
|   |   IF run_def->DI.DI_val[di].PD.doppler_aug_sel != c_spd_none_augment THEN
|   |   |   t_cval_run_naug_bins[di] = run_def->DI.DI_val[di].PD.PD_FFT_len * \
|   |   |   run_def->DI.DI_val[di].PD.unambig_PRF / run_def->radar_proc.nominal_PRF
|   |   ENDIF
```

```
    |   ELSE
    |   |   t_cval_run_nbins[di] = NUM_PULSES_PROC - t_cval_run_num_pulses_lost
    |   |   t_cval_run_naug_bins[di] = NUM_PULSES_PROC - t_cval_run_num_pulses_lost
    |   ENDIF
    ENDLOOP
*/
}

/* end of cval_run_local.c */
```

```
/*%%============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_run_config.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---  -----------  ------  ------------------------------------------------
n/a  15-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_run_config

PURPOSE:        This function validates the system configuration for the simulation run data.


 INPUT                                              SOURCE
 -----                                              ------
 run_def                                            calling routine
 sat_def                                            calling routine
 num_msg                                            calling routine
 cur_log_entry                                      calling routine
 t_cval_run_num_blankers                            CVAL global data
 t_cval_run_num_ph_ctr_per_grp                      CVAL global data
 t_cval_run_DPCA_mode                               CVAL global data
 t_cval_run_num_DPCA_grps                           CVAL global data

 OUTPUT                                             DESTINATION
 ------                                             -----------
 return_code                                        calling routine
 num_msg                                            calling routine
 cur_log_entry                                      calling routine

 CREATION DATE:  June 15 1993

 ============================================================================*/
 #include "cval.h"

 #define MONOPULSE_ENABLE    run_def->general_config.monopulse_enable
 #define NUM_RG_DOP_BLANKERS run_def->general_config.num_rg_dop_blankers
```

```
#define SC_ENABLE          run_def->radar_proc.SC_enable
#define MOTION_COMP        sat_def->antenna_model.main_rxfe.motion_comp

f_cval_run_config(
    t_csid_run_def *run_def,
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
    initialize the return_code to c_spd_success
    IF ((t_cval_run_num_ph_ctr_per_grp < 2) && (t_cval_run_DPCA_mode != c_spd_DPCA_none)) THEN
    |   Use f_cval_log_msg with index to message indicating\
    |   "ERROR DPCA requires two or more phase centres per group"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
    IF ((MONOPULSE_ENABLE == c_spd_true) &&\
    |   ((t_cval_run_num_ph_ctr_per_grp < 3) && (t_cval_run_DPCA_mode == c_spd_ASAR_DPCA) ||\
    (t_cval_run_num_DPCA_grps < 2) && (t_cval_run_num_ph_ctr_per_grp < 2))) THEN
    |   Use f_cval_log_msg with index to message indicating\
    |   "ERROR Monopulse requires more phase centres"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
    IF (((MONOPULSE_ENABLE == c_spd_true) && (NUM_RG_DOP_BLANKERS > 0)) ||\
    ((t_cval_run_DPCA_mode == c_spd_ASAR_DPCA) && (NUM_RG_DOP_BLANKERS > 1))) THEN
    |   Use f_cval_log_msg with index to message indicating\
    |   "ERROR number of range Doppler blankers exceeds maximum allowed"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF

    IF (t_cval_run_num_blankers > sat_def->antenna_model.aux_count) THEN
    |   Use f_cval_log_msg with index to message indicating\
    |   "WARNING number of blankers exceeds number of auxiliaries"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
    IF ((t_cval_run_DPCA_mode == c_spd_MASR_DPCA) && (MONOPULSE_ENABLE == c_spd_true) &&\
    (t_cval_run_num_DPCA_grps < 2) && (t_cval_run_num_ph_ctr_per_grp < 3)) THEN
    |   Use f_cval_log_msg with index to message indicating\
    |   "ERROR MASR requires more phase centres to operate with monopulse"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
    IF ((MOTION_COMP == c_spd_true) && (SC_ENABLE == c_spd_true)) THEN
    |   Use f_cval_log_msg with index to message indicating\
    |   "ERROR sampling correction not compatible with PRF-adjusted system"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
<---return the return_code
*/
}

/* end of cval_run_config.c */
```

```
/**%%========================================================================

 SPD - Signal Processing Demonstrator Computer System

 (C) Copyright Her Majesty the Queen in Right of Canada as
 represented by the Minister of National Defence, 1992

 File Name:      cval_run_beam.c

 Identification: [@(#)]

 SCCS Id:        %W% %E%

 Responsible Organization:

 Raytheon Canada Limited

 Designer:       Chris Candler

 Change History:

 SCR    Date        Author   Change made
 ---  -----------  ------  ---------------------------------------------------
 n/a  15-JUN-1993  fcc     Initial Release

 CSU_NAME:        CVAL

 FUNCTION:        f_cval_run_beam

 PURPOSE:         This function validates the main beam configuration for the run-time
 consistency check.


 INPUT                                          SOURCE
 -----                                          ------
 run_def                                        calling routine
 sat_def                                        calling routine
 num_msg                                        calling routine
 cur_log_entry                                  calling routine
 t_cval_run_num_mono                            CVAL global data
 t_cval_run_npcu                                CVAL global data
 t_cval_run_max_nul_chan                        CVAL global data
 t_cval_run_ndoft                               CVAL global data
 t_cval_run_num_channels                        CVAL global data

 OUTPUT                                         DESTINATION
 ------                                         -----------
 num_msg                                        calling routine
 cur_log_entry                                  calling routine
 return_code                                    calling routine

 CREATION DATE:  June 15 1993

 ==========================================================================*/
 #include "cval.h"

```

```
#define NUM_NULLING_CHAN_PER_PH_CTR run_def->ECCM_config.pc_ctr_nulling[syspc].num_nulling_chan
#define PH_CTR_NULLING_CHAN         run_def->ECCM_config.pc_ctr_nulling[syspc].nulling_chan[ch]
#define NEXT_PH_CTR_NULLING_CHAN    run_def->ECCM_config.pc_ctr_nulling[syspc].nulling_chan[dh]

f_cval_run_beam(
    t_csid_run_def *run_def,
    t_spd_int32    *num_msg,
    char            **cur_log_entry
)
{
/*|
    initialize the return_code to c_spd_success
    FOR sys = 0 to t_cval_run_num_mono - 1
    LOOP
    |   num_nul_chan_per_sys = npcu - 1
    |   FOR pc = 0 TO t_cval_run_npcu - 1
    |   LOOP
    |   |   syspc = sys * t_cval_run_num_ph_ctr_per_grp + pc
    |   |   num_nul_chan_per_sys += NUM_NULLING_CHAN_PER_PH_CTR
    |   |   IF (NUM_NULLING_CHAN_PER_PH_CTR > t_cval_run_ndoft) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "WARNING number of nulling channels for system (sys) phase centre (pc)\
    |   |   |   exceeds number of dof"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF (NUM_NULLING_CHAN_PER_PH_CTR > t_cval_run_max_nul_chan) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR number of nulling channels for system (sys) phase centre (pc)\
    |   |   |   exceeds number allowed"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   FOR ch = 0 to NUM_NULLING_CHAN_PER_PH_CTR - 1
    |   |   LOOP
    |   |   |   IF PH_CTR_NULLING_CHAN > t_cval_run_num_channels THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "ERROR nulling channel (ch) for system (sys) phase centre (pc)\
    |   |   |   |   exceeds number of channels"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   |   IF PH_CTR_NULLING_CHAN <= t_cval_run_num_ph_ctrs THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "WARNING (nulling channel (ch) for system (sys) phase centre (pc)\
    |   |   |   |   is a main bean"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   |   FOR dh = ch + 1 to NUM_NULLING_CHAN_PER_PH_CTR - 1
    |   |   |   LOOP
    |   |   |   |   IF PH_CTR_NULLING_CHAN == NEXT_PH_CTR_NULLING_CHAN THEN
```

```
 |   |   |   |   |   Use f_cval_log_msg with index to message indicating\
 |   |   |   |   |   "WARNING nulling channels (ch) (dh) for system (sys) phase\
 |   |   |   |   |   centre (pc) are the same"
 |   |   |   |   |   IF there was an error writing the message THEN
 |   |   |   |   |   |   set the return_code to c_spd_failure
 |   |   |   |   |   ENDIF
 |   |   |   |   ENDIF
 |   |   |   ENDLOOP
 |   |   ENDLOOP
 |   ENDLOOP
 |   IF num_nul_chan_per_sys > t_cval_run_max_sys_size THEN
 |   |   Use f_cval_log_msg with index to message indicating\
 |   |   "ERROR number of nulling channels for system (sys) exceeds number allowed"
 |   |   IF there was an error writing the message THEN
 |   |   |   set the return_code to c_spd_failure
 |   |   ENDIF
 |   ENDIF
ENDLOOP
<---return the return_code
*/
}

/* end of cval_run_beam.c */
```

```
/*%%============= ================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_run_blanker.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---  -----------  ------  ------------------------------------------------
n/a  15-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_run_blanker

PURPOSE:        This function validates the blanker configuration for the the run-time
consistency check.


INPUT                                              SOURCE
-----                                              ------
run_def                                            calling routine
sat_def                                            calling routine
num_msg                                            calling routine
cur_log_entry                                      calling routine
t_cval_run_ndoft                                   CVAL global data
t_cval_run_num_channels                            CVAL global data
t_cval_run_num_ph_ctrs                             CVAL global data
t_cval_run_max_nul_chan                            CVAL global data

OUTPUT                                             DESTINATION
------                                             -----------
num_msg                                            calling routine
cur_log_entry                                      calling routine
return_code                                        calling routine

CREATION DATE:  June 15 1993

=================================================================================*/
#include "cval.h"

#define NUM_NULLING_CHAN_PER_PH_CTR run_def->ECCM_config.pc_ctr_nulling[syspc].num_nulling_chan
```

```
#define PH_CTR_NULLING_CHAN          run_def->ECCM_config.pc_ctr_nulling[syspc].nulling_chan[ch]
#define NEXT_PH_CTR_NULLING_CHAN     run_def->ECCM_config.pc_ctr_nulling[syspc].nulling_chan[dh]
#define BLANKER_CHAN                 run_def->ECCM_config.blanker_chan[bl]
#define NUM_NULLING_CHAN_PER_BLKR    run_def->ECCM_config.blanker_nulling[bl].num_nulling_chan
#define BLANKER_NULLING_CHAN         run_def->ECCM_config.blanker_nulling[bl].nulling_chan[ch]
#define NEXT_BLANKER_NULLING_CHAN    run_def->ECCM_config.blanker_nulling[bl].nulling_chan[dh]
#define NUM_PRE_DOP_BLANKERS         run_def->ECCM_proc.num_pre_dop_blankers
#define PREDOP_BLK_SLOPE_THRSHD      run_def->ECCM_proc.predop_blk_slope_thrshd
#define PREDOP_BLK_NOISE_THRSHD      run_def->ECCM_proc.predop_blk_noise_thrshd
#define NUM_RG_DOP_BLANKERS          run_def->general_config.num_rg_dop_blankers
#define RGDOP_BLK_SLOPE_THRSHD       run_def->radar_proc.rgdop_blk_slope_thrshd
#define RGDOP_BLK_NOISE_THRSHD       run_def->radar_proc.rgdop_blk_noise_thrshd

f_cval_run_blanker(
    t_csid_run_def *run_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
    initialize the return_code to c_spd_success
    FOR id = 0 TO t_cval_run_num_ph_ctrs - 1
    LOOP
    |   invaux[id] = 0
    ENDLOOP
    id = t_cval_run_num_ph_ctrs + 1
    FOR aux = 0 TO sat_def->antenna_model.aux_count - 1
    LOOP
    |   aux_chan_id[aux] = id
    |   invaux[id] = aux
    |   id = id + 1
    |   FOR j = 0 TO sat_def->antenna_model.aux_antenna[aux].tap_count - 1
    |   LOOP
    |   |   tap_chan_id[aux][j] = id
    |   |   invaux[id] = aux
    |   |   id = id + 1
    |   ENDLOOP
    ENDLOOP
    FOR  bl = 0 to t_cval_run_num_blankers - 1
    LOOP
    |   IF BLANKER_CHAN <= t_cval_num_ph_ctrs THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "WARNING blanker (bl) is a main beam"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   IF NUM_NULLING_CHAN_PER_BLKR > t_cval_run_max_nul_chan THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "ERROR number of nulling channels for blanker (bl) exceeds number allowed"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   FOR ch = 0 TO NUM_NULLING_CHAN_PER_BLKR - 1
    |   LOOP
|   |   IF BLANKER_NULLING_CHAN > t_cval_run_num_channels THEN
|   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   "ERROR nulling channel (ch) for blanker (bl) exceeds number of channels"
|   |   |   IF there was an error writing the message THEN
|   |   |   |   set the return_code to c_spd_failure
|   |   |   ENDIF
|   |   ENDIF
|   |   IF BLANKER_NULLING_CHAN <= t_cval_run_num_ph_ctrs THEN
|   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   "WARNING nulling channel (ch) for blanker (bl) is a main beam"
|   |   |   IF there was an error writing the message THEN
|   |   |   |   set the return_code to c_spd_failure
|   |   |   ENDIF
|   |   ENDIF
|   |   FOR dh = ch + 1 TO NUM_NULLING_CHAN_PER_BLKR - 1
|   |   LOOP
|   |   |   IF BLANKER_NULLING_CHAN == NEXT_BLANKER_NULLING_CHAN THEN
|   |   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   |   "WARNING nulling channels (ch) (dh) for blanker (bl) are the same"
|   |   |   |   IF there was an error writing the message THEN
|   |   |   |   |   set the return_code to c_spd_failure
|   |   |   |   ENDIF
|   |   |   ENDIF
|   |   ENDLOOP
|   ENDLOOP
|   IF BLANKER_CHAN > t_cval_run_num_channels THEN
|   |   Use f_cval_log_msg with index to message indicating\
|   |   "ERROR blanker channel (bl) exceeds number of channels"
|   |   IF there was an error writing the message THEN
|   |   |   set the return_code to c_spd_failure
|   |   ENDIF
|   ELSE
|   |   aux_no = invaux[BLANKER_CHAN]
|   |   IF (NUM_NULLING_CHAN_PER_BLKR >\
|   |   t_cval_run_ndoft - 1 - sat_def->antenna_model.aux_antenna[aux_no].tap_count) THEN
|   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   "WARNING number of nulling channels for blanker (bl) exceeds number of dof"
|   |   |   IF there was an error writing the message THEN
|   |   |   |   set the return_code to c_spd_failure
|   |   |   ENDIF
|   |   ENDIF
|   |   FOR ch = 0 to NUM_NULLING_CHAN_PER_BLKR - 1
|   |   LOOP
|   |   |   IF BLANKER_NULLING_CHAN == aux_chan_id[aux_no] THEN
|   |   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   |   "WARNING blanker (bl) nulled by itself (ch)"
|   |   |   |   IF there was an error writing the message THEN
|   |   |   |   |   set the return_code to c_spd_failure
|   |   |   |   ENDIF
|   |   |   ENDIF
|   |   |   FOR j = 0 to sat_def->antenna_model.aux_antenna[aux_no].tap_count - 1
|   |   |   LOOP
|   |   |   |   IF BLANKER_NULLING_CHAN == tap_chan_id[aux_no][j] THEN
|   |   |   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   |   |   "WARNING blanker (bl) nulled by itself (ch)"
|   |   |   |   |   IF there was an error writing the message THEN
```

```
    |   |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   |   ENDIF
    |   |   |   |   ENDIF
    |   |   |   ENDLOOP
    |   |   ENDLOOP
    |   ENDIF
    ENDLOOP
    // now check the blanker thresholds
    IF ((NUM_PRE_DOP_BLANKERS > 0) && (PREDOP_BLK_SLOPE_THRSHD == 0) &&\
    (PREDOP_BLK_NOISE_THRSHD == 0)) THEN
    |   Use f_cval_log_msg with index to message indicating\
    |   "ERROR both pre-Doppler blanking thresholds are zero"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
    IF ((NUM_RG_DOP_BLANKERS > 0) && (RGDOP_BLK_SLOPE_THRSHD == 0) &&\
    (RGDOP_BLK_NOISE_THRSHD == 0)) THEN
    |   Use f_cval_log_msg with index to message indicating\
    |   "ERROR both post-Doppler blanking thresholds are zero"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
<---return the return_code
*/
}

/* end of cval_run_blanker.c */
```

D-245

```
/*%%=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:       cval_run_norm_equ.c

Identification: [@(#)]

SCCS Id:         %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:        Chris Candler

Change History:

SCR   Date         Author   Change made
---  -----------  ------  ------------------------------------------------
n/a  15-JUN-1993  fcc     Initial Release

CSU_NAME:        CVAL

FUNCTION:        f_cval_run_norm_equ

PURPOSE:         This function validates the normal equations for the run-time consistency check.


INPUT                                              SOURCE
-----                                              ------
run_def                                            calling routine
sat_def                                            calling routine
num_msg                                            calling routine
cur_log_entry                                      calling routine
t_cval_run_num_mono                                CVAL global data
t_cval_run_npcu                                    CVAL global data
t_cval_run_num_channels                            CVAL global data
t_cval_run_num_blankers                            CVAL global data
t_cval_run_samps_pri[][]                           CVAL global data
t_cval_run_bytes_per_samp                          CVAL global data
t_cval_run_M[]                                     CVAL global data
t_cval_run_avg_num_snaps_per_upd                   CVAL global data
t_cval_run_num_weight_vectors[][]                  CVAL global data
t_cval_monopulse_enable                            CVAL global data


OUTPUT                                             DESTINATION
------                                             -----------
num_msg                                            calling routine
cur_log_entry                                      calling routine
return_code                                        calling routine

```

```
**************************************************************************/
#include "cval.h"

#define NUM_NULLING_CHAN_PER_BLKR    run_def->ECCM_config.blanker_nulling[bl].num_nulling_chan
#define SNAPSHOT_SMOOTH_PERIOD       run_def->ECCM_proc.snapshot_smooth_period
#define NUM_SNAPSHOTS                run_def->ECCM_proc.num_snapshots
#define RXFE_SAMPLING_RATE           sat_def->antenna_model.main_rxfe.sampling_rate
#define NUM_UPDATE_INTS              run_def->ECCM_proc.num_update_ints
#define SNAPSHOT_EDIT                run_def->ECCM_proc.snapshot_edit
#define NUM_SNAPSHOTS_REJECTED       run_def->ECCM_proc.num_snapshots_rejected
#define MOTION_COMP                  sat_def->antenna_model.main_rxfe.motion_comp
#define PULSE_PER_CPI                sat_def->DI.DI_val[di].NCI.NCI_val[nci].pulse_per_CPI

f_cval_run_norm_equ(
    t_csid_run_def *run_def,
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
    initialize the return_code to c_spd_success
    num_sys = t_cval_run_num_mono + t_cval_run_num_blankers
    IF t_cval_run_npcu == 1 THEN
    |   t_cval_run_M[1] = run_def->ECCM_config.pc_ctr_nulling[0].num_nulling_chan
    |   IF t_cval_run_monopulse_enable == c_spd_true THEN
    |   |   t_cval_run_M[2] = run_def->ECCM_config.pc_ctr_nulling[1].num_nulling_chan
    |   ENDIF
    ELSEIF t_cval_run_npcu == 2 THEN
    |   t_cval_run_M[1] = run_def->ECCM_config.pc_ctr_nulling[0].num_nulling_chan +\
    |   run_def->ECCM_config.pc_ctr_nulling[1].num_nulling_chan + 1
    |   IF t_cval_run_monopulse_enable == c_spd_true THEN
    |   |   t_cval_run_M[2] = run_def->ECCM_config.pc_ctr_nulling[2].num_nulling_chan +\
    |   |   run_def->ECCM_config.pc_ctr_nulling[3].num_nulling_chan + 1
    |   ENDIF
    ELSEIF t_cval_run_npcu == 2 THEN
    |   t_cval_run_M[1] = run_def->ECCM_config.pc_ctr_nulling[0].num_nulling_chan +\
    |   run_def->ECCM_config.pc_ctr_nulling[1].num_nulling_chan + \
    |   run_def->ECCM_config.pc_ctr_nulling[2].num_nulling_chan + 2
    |   IF t_cval_run_monopulse_enable == c_spd_true THEN
    |   |   t_cval_run_M[2] = run_def->ECCM_config.pc_ctr_nulling[3].num_nulling_chan +\
    |   |   run_def->ECCM_config.pc_ctr_nulling[4].num_nulling_chan +\
    |   |   run_def->ECCM_config.pc_ctr_nulling[5].num_nulling_chan + 2
    |   ENDIF
    ENDIF
    FOR bl = 0 to t_cval_run_num_blankers - 1
    LOOP
    |   t_cval_run_M[bl + t_cval_run_num_mono] = NUM_NULLING_CHAN_PER_BLKR
    ENDLOOP
    max_M = 0
    FOR sys = 0 to num_sys - 1
    LOOP
    |   max_M = max(t_cval_run_M[sys], max_M)
    ENDLOOP
```

```
weight_update_allowed = 500E03
TPM_size = 32E06
IF (max_M > 0) THEN
|   samples_per_smooth_period = 1.0E03 * SNAPSHOT_SMOOTH_PERIOD * RXFE_SAMPLING_RATE
|   IF (samples_per_smooth_period < max_M) THEN
|   |   Use f_cval_log_msg with index to message indicating\
|   |   "ERROR smoothing period too short"
|   |   IF there was an error writing the message THEN
|   |   |   set the return_code to c_spd_failure
|   |   ENDIF
|   ENDIF
|   IF (NUM_SNAPSHOTS < max_M) THEN
|   |   Use f_cval_log_msg with index to message indicating "ERROR not enough snapshots"
|   |   IF there was an error writing the message THEN
|   |   |   set the return_code to c_spd_failure
|   |   ENDIF
|   ENDIF
|   IF (NUM_SNAPSHOTS > samples_per_smooth_period) THEN
|   |   Use f_cval_log_msg with index to message indicating "ERROR too many snapshots"
|   |   IF there was an error writing the message THEN
|   |   |   set the return_code to c_spd_failure
|   |   ENDIF
|   ENDIF
|   IF (NUM_UPDATE_INTS > NUM_SNAPSHOTS) THEN
|   |   Use f_cval_log_msg with index to message indicating\
|   |   "ERROR too many update intervals smoothed "
|   |   IF there was an error writing the message THEN
|   |   |   set the return_code to c_spd_failure
|   |   ENDIF
|   ENDIF
|   IF (SNAPSHOT_EDIT != c_spd_none) THEN
|   |   t_cval_run_avg_num_snaps_per_upd =NUM_SNAPSHOTS/NUM_UPDATE_INTS-NUM_SNAPSHOTS_REJECTED
|   |   // t_cval_run_avg_num_snaps_per_upd is floating point
|   |   IF ((t_cval_run_avg_num_snaps_per_upd * NUM_UPDATE_INTS < max_M) &&\
|   |   (NUM_SNAPSHOTS >= max_M)) THEN
|   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   "ERROR too many snapshots rejected"
|   |   |   IF there was an error writing the message THEN
|   |   |   |   set the return_code to c_spd_failure
|   |   |   ENDIF
|   |   ENDIF
|   ELSE
|   |   t_cval_run_avg_num_snaps_per_upd = NUM_SNAPSHOTS / NUM_UPDATE_INTS
|   ENDIF
|   weight_update_bytes = ((NUM_SNAPSHOTS * t_cval_run_num_channels) +\
|   (max_M * max_M + max_M) * (3 + NUM_UPDATE_INTS)) * 2 * sizeof(t_spd_double)
|   IF (weight_update_bytes > weight_update_allowed) THEN
|   |   Use f_cval_log_msg with index to message indicating\
|   |   "ERROR insufficient memory for weight computation, reduce system\
|   |   size or number of smoothed intervals"
|   |   IF there was an error writing the message THEN
|   |   |   set the return_code to c_spd_failure
|   |   ENDIF
|   ENDIF
|   FOR di = 0 TO run_def->DI.DI_len - 1
|   LOOP
```

2-69

```
    |   |   FOR nci = 0 TO run_def->DI.DI_val[di].NCI.NCI_len - 1
    |   |   LOOP
    |   |   |   IF (t_cval_run_CPI_time[d][n] < SNAPSHOT_SMOOTH_PERIOD) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "WARNING processed CPIs shorter than smoothing period (di,nci)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   |   FOR cpi = 0 TO run_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_len - 1
    |   |   |   LOOP
    |   |   |   |   IF (MOTION_COMP == c_spd_false) THEN
    |   |   |   |   |   extra_samps = f_RSS_Determine_Number_of_Extra_Samples()
    |   |   |   |   ELSE
    |   |   |   |   |   extra_samps = 0
    |   |   |   |   ENDIF
    |   |   |   |   cpi_data = (PULSE_PER_CPI * t_cval_run_samps_pri[d][n] + extra_samps) *\
    |   |   |   |   t_cval_run_bytes_per_samp
    |   |   |   |   IF (t_cval_run_num_weight_vectors[di][nci] * max_M * 2 * sizeof(t_spd_real)\
    |   |   |   |   + cpi_data > TPM_size) THEN
    |   |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   |   "ERROR weights unable to fit in TPM, reduce system size or number\
    |   |   |   |   |   of updates"
    |   |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   |   ENDIF
    |   |   |   |   ENDIF
    |   |   |   ENDLOOP
    |   |   ENDLOOP
    |   ENDLOOP
    ENDIF
<---return the return_code
*/
}

/* end of cval_run_norm_equ.c */
```

```
/*%%============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_run_pulse_compr.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR    Date        Author   Change made
---  -----------  ------  ------------------------------------------------
n/a  15-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_run_pulse_compr

PURPOSE:        This function validates the pulse compression configuration for the the run-time
consistency check.


INPUT                                                   SOURCE
-----                                                   ------
run_def                                                 calling routine
num_msg                                                 calling routine
cur_log_entry                                           calling routine
t_cval_run_naug_bins                                    CVAL global data
t_cval_run_nbins                                        CVAL global data
t_cval_run_samps_pri                                    CVAL global data
t_cval_run_num_dop                                      CVAL global data

OUTPUT                                                  DESTINATION
------                                                  -----------
num_msg                                                 calling routine
cur_log_entry                                           calling routine
return_code                                             calling routine

CREATION DATE:  June 15 1993

==============================================================================*/
#include "cval.h"

f_cval_run_pulse_compr(
    t_csid_run_def *run_def,
    t_spd_int32     *num_msg,
    char            **cur_log_entry
)
(
/*|
    initialize the return_code to c_spd_success
    IF (run_def->radar_proc.pulse_compr_enable == c_spd_true) &&\
    run_def->radar_proc.doppler_comp_enable == c_spd_true) THEN
    |   Use f_cval_log_msg with index to message indicating\
    |   "ERROR Doppler compensation not applicable without pulse compression"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
    npe_pc = 36
    max_pc_words = 150E03
    FOR di = 0 to run_def->DI.DI_len - 1
    LOOP
    |   IF (run_def->DI.DI_val[di].PD.doppler_aug_sel == c_spd_PC_augment)  THEN
    |   |   nvpe = ceil(t_cval_run_naug_bins[di] / npe_pc)
    |   ELSE
    |   |   nvpe = ceil(t_cval_run_nbins[di] / npe_pc)
    |   ENDIF
    |   FOR nci = 0 TO run_def->DI.DI_val[di].NCI.NCI_len - 1
    |   LOOP
    |   |   nwords = nvpe * t_cval_run_num_dop * t_cval_run_samps_pri[d]
    |   |   IF nwords > max_pc_words THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR insufficient memory for pulse compression (di,nci)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   ENDLOOP
    ENDLOOP
<---return the return_code
*/
}

/* end of cval_run_pulse_compr.c */
```

D-29

```
/*%%=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_run_unambig_prf.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date         Author   Change made
---   -----------  ------   ------------------------------------------------
n/a   15-JUN-1993  fcc      Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_run_unambig_prf

PURPOSE:        This function checks if the unambiguous PRF field is in the correct range
for the run-time consistency check.


                                                SOURCE
INPUT                                           ------
-----                                           calling routine
run_def                                         calling routine
num_msg                                         calling routine
cur_log_entry                                   CVAL global data
t_cval_run_PRF_min

                                                DESTINATION
OUTPUT                                          -----------
------                                          calling routine
num_msg                                         calling routine
cur_log_entry                                   calling routine
return_code

CREATION DATE:  June 15 1993

=========================================================================*/
#include "cval.h"


f_cval_run_unambig_prf(
     t_csid_run_def *run_def,
     t_spd_int32    *num_msg,
     char           **cur_log_entry
```

D-[illegible]

```
}
{
/*|
    initialize the return_code to c_spd_success
    IF (run_def->DI.DI_val[di].PD.doppler_aug_sel == c_spd_none_augment) THEN
    |   FOR di = 0 to run_def->DI.DI_len -1
    |   LOOP
    |   |   IF (run_def->DI.DI_val[di].PD.unambig_PRF < t_cval_run_PRF_min[di]) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR unambiguous PRF not large enough (di)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF (run_def->DI.DI_val[di].PD.unambig_PRF > 2 * t_cval_run_PRF_min[di]) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR unambiguous PRF too large (di)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   ENDLOOP
    ENDIF
<---return the return_code
*/
}

/* end of cval_run_unambig_prf.c */
```

D-[illegible]

```
/*%%===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_run_dpca.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date         Author    Change made
---  -----------  ------  ----------------------------------------------
n/a  15-JUN-1993  fcc      Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_run_dpca

PURPOSE:        This function validates the Doppler and ASAR DPCA for the run-time
consistency check.


INPUT                                                    SOURCE
-----                                                    ------
run_def                                                  calling routine
sat_def                                                  calling routine
num_msg                                                  calling routine
cur_log_entry                                            calling routine
t_cval_run_DPCA_mode                                     CVAL global data
t_cval_run_samps_pri                                     CVAL global data

OUTPUT                                                   DESTINATION
------                                                   -----------
num_msg                                                  calling routine
cur_log_entry                                            calling routine
return_code                                              calling routine

CREATION DATE:  June 15 1993

===========================================================================*/
#include "cval.h"

#define START_RNG_GATE     run_def->DI.DI_val[di].ASAR_DPCA.start_rng_gate
#define NUM_CC_RNG_GATES   run_def->DI.DI_val[di].ASAR_DPCA.num_cc_rng_gates
#define STEP_RNG_GATE      run_def->DI.DI_val[di].ASAR_DPCA.step_rng_gate
```

```
#define NUM_DPCA_CELLS    run_def->DI.DI_val[di].ASAR_DPCA.num_DPCA_cells
#define PD_FFT_LENGTH     run_def->DI.DI_val[di].PD.PD_FFT_len
#define PRI_MULTIPLIER    sat_def->antenna_model.main_antenna.CFA.PRI_multiplier
#define NUM_PULSES_PROC   run_def->DI.DI_val[di].NCI.NCI_val[nci].num_pulses_proc
#define NUM_PULSES_TX     run_def->DI.DI_val[di].NCI.NCI_val[nci].num_pulses_tx
#define RNG_GATE_TYPE     run_def->DI.DI_val[di].ASAR_DPCA.rng_gate_type

f_cval_run_dpca(
    t_csid_run_def *run_def,
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
      initialize the return_code to c_spd_success
      IF (t_cval_run_DPCA_mode == c_spd_ASAR_DPCA) &&\
      (run_def->radar_proc.PD_enable == c_spd_true) THEN
      |  Use f_cval_log_msg with index to message indicating\
      |  "ERROR ASAR DPCA must have pulse-Doppler enabled"
      |  IF there was an error writing the message THEN
      |  |  set the return_code to c_spd_failure
      |  ENDIF
      ENDIF
      FOR di = 0 to run_def->DI.DI_len - 1
      LOOP
      |  IF (t_cval_run_DPCA_mode == c_spd_ASAR_DPCA) &&\
      |  (NUM_DPCA_CELLS > PD_FFT_LENGTH/PRI_MULTIPLIER) THEN
      |  |  Use f_cval_log_msg with index to message indicating\
      |  |  "ERROR too many ASAR DPCA Doppler cells (di)"
      |  |  IF there was an error writing the message THEN
      |  |  |  set the return_code to c_spd_failure
      |  |  ENDIF
      |  ENDIF
      |  FOR nci = 0 to run_def->DI.DI_val[di].NCI.NCI_len[d] - 1
      |  LOOP
      |  |  IF (NUM_PULSES_PROC > NUM_PULSES_TX) THEN
      |  |  |  Use f_cval_log_msg with index to message indicating\
      |  |  |  "ERROR number of pulses processed more than number transmitted (di,nci)"
      |  |  |  IF there was an error writing the message THEN
      |  |  |  |  set the return_code to c_spd_failure
      |  |  |  ENDIF
      |  |  ENDIF
      |  |  IF (PD_FFT_LENGTH < NUM_PULSES_PROC - num_pulses_lost) &&\
      |  |  (PD_ENABLE == c_spd_true) THEN
      |  |  |  Use f_cval_log_msg with index to message indicating\
      |  |  |  "ERROR FFT length less than number of pulses processed\
      |  |  |  (less DPCA delay) (di,nci) "
      |  |  |  IF there was an error writing the message THEN
      |  |  |  |  set the return_code to c_spd_failure
      |  |  |  ENDIF
      |  |  ENDIF
      |  |  IF (t_cval_run_DPCA_mode == c_spd_ASAR_DPCA) THEN
      |  |  |  IF (NUM_CC_RNG_GATES > t_cval_run_samps_pri[di][nci]) THEN
      |  |  |  |  Use f_cval_log_msg with index to message indicating\
      |  |  |  |  "ERROR too many ASAR DPCA range gates (di,nci) "
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   |   last_gate = START_RNG_GATE + (NUM_CC_RNG_GATES - 1) * STEP_RNG_GATE
    |   |   |   IF ((RNG_GATE_TYPE == c_spd_start_gate_step) &&\
    |   |   |   (last_gate > t_cval_run_samps_pri[di][nci])) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "ERROR ASAR DPCA start gate and step goes beyond ambiguous\
    |   |   |   |   range (di,nci) "
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   ENDIF
    |   ENDLOOP
    ENDLOOP
<---return the return_code
*/
}

/* end of cval_run_dpca.c */
```

```
/*%%=========================================================================
SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_run_non_coh.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR    Date        Author    Change made
---  -----------  ------  -------------------------------------------------
n/a  15-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_run_non_coh

PURPOSE:        This function validates the non coherent integration for the run-time
consistency check.


INPUT                                            SOURCE
-----                                            ------
run_def                                          calling routine
sat_def                                          calling routine
num_msg                                          calling routine
cur_log_entry                                    calling routine
t_cval_run_num_mono                              CVAL global data
t_cval_run_naug_bins                             CVAL global data

OUTPUT                                           DESTINATION
------                                           -----------
num_msg                                          calling routine
cur_log_entry                                    calling routine
return_code                                      calling routine

CREATION DATE:  June 15 1993

===========================================================================*/
#include "cval.h"
#define NUM_BURSTS_INTEG run_def->DI.DI_val[di].NCI.NCI_val[nci].num_bursts_integ
#define CPI_LEN          run_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_len
#define OS_NCI_RANK      run_def->DI.DI_val[di].OS_NCI_rank

```

```
f_cval_run_non_coh(
    t_csid_run_def *run_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
    initialize the return_code to c_spd_success
    npe_nci = 8
    max_nci_words = 190E03
    FOR di = 0 TO run_def->DI.DI_len - 1
    LOOP
    |   nvpe = ceil(1.25 * ceil(t_cval_run_naug_bins[d] / npe_nci))
    |   FOR nci = 0 to run_def->DI.DI_val[di].NCI.NCI_len - 1
    |   LOOP
    |   |   IF NUM_BURSTS_INTEG > CPI_LEN THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR number of bursts integrated more than transmitted (di,nci)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF (run_def->radar_proc.integ_method == c_spd_ordered_statistic) &&\
    |   |   (OS_NCI_RANK > NUM_BURSTS_INTEG) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR OS rank greater than number of bursts integrated (di,nci)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF (run_def->radar_proc.integ_method == c_spd_ordered_statistic) THEN
    |   |   |   nbuf = NUM_BURSTS_INTEG - OS_NCI_RANK + 1
    |   |   |   nwords = nbuf * nvpe * t_cval_run_num_mono * t_cval_run_samps_pri[di][nci]
    |   |   ELSE
    |   |   |   nwords = nvpe * t_cval_run_num_mono * t_cval_run_samps_pri[di][nci]
    |   |   ENDIF
    |   |   IF (nwords > max_nci_words) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR insufficient memory for non-coherent integration (di,nci)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   ENDLOOP
    ENDLOOP
<---return the return_code
*/
}

/* end of cval_run_non_coh.c */
```

```
/*%%============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_run_cfar.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date         Author   Change made
---   -----------  ------   ---------------------------------------------
n/a   15-JUN-1993  fcc      Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_run_cfar

PURPOSE:        This function validates the CFAR parameters for the run-time
consistency check.


INPUT                                             SOURCE
-----                                             ------
run_def                                           calling routine
num_msg                                           calling routine
cur_log_entry                                     calling routine
t_cval_run_naug_bins                              CVAL global data
t_cval_run_PRF_max                                CVAL global data
t_cval_run_samps_pri                              CVAL global data

OUTPUT                                            DESTINATION
------                                            -----------
num_msg                                           calling routine
cur_log_entry                                     calling routine
return_code                                       calling routine

CREATION DATE:  June 15 1993

==============================================================================*/
#include "cval.h"
#define DOPPLER_WIDTH       run_def->DI.DI_val[di].detect_thrshd.doppler_width
#define RANGE_WIDTH         run_def->DI.DI_val[di].detect_thrshd.range_width
#define GUARD_DOPPLER_WIDTH run_def->DI.DI_val[di].detect_thrshd.guard_doppler_width
#define GUARD_RANGE_WIDTH   run_def->DI.DI_val[di].detect_thrshd.guard_range_width
```

D-699

```

f_cval_run_cfar(
    t_csid_run_def *run_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
(
/*|
    initialize the return_code to c_spd_success
    npe_det = 4
    max_cfar_words = 100E03
    IF run_def->radar_proc.detect_type == c_spd_fixed_threshold THEN
    |   FOR di = 0 TO run_def->DI.DI_len - 1
    |   LOOP
    |   |   IF  DOPPLER_WIDTH > t_cval_run_naug_bins[di] THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR CFAR Doppler width too great (di)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF  RANGE_WIDTH > round(RXFE_SAMPLING_RATE / t_cval_run_PRF_max[di] THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR CFAR range width too great (di)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF  ((RANGE_WIDTH == 1) && (DOPPLER_WIDTH == 1)) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR CFAR range width and Doppler width cannot both be 1 (di)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF  (GUARD_DOPPLER_WIDTH > DOPPLER_WIDTH) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR CFAR guard Doppler width too great (di)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF  ((GUARD_RANGE_WIDTH == RANGE_WIDTH) && (GUARD_DOPPLER_WIDTH == DOPPLER_WIDTH) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR CFAR guard range width and guard Doppler width cannot both be\
    |   |   |   maximum (di)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   num_cells = DOPPLER_WIDTH * RANGE_WIDTH - GUARD_DOPPLER_WIDTH * GUARD_RANGE_WIDTH
    |   |   IF (OS_CFAR_RANK > num_cells) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR CFAR OS rank exceeds number of reference cells (di)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   nvpe = ceil(t_cval_run_naug_bins[di] / npe_det) + DOPPLER_WIDTH
    |   |   FOR nci = 0 TO run_def->DI.DI_val[di].NCI.NCI_len - 1
    |   |   LOOP
    |   |   |   nwords = nvpe * t_cval_run_samps_pri[di][nci]
    |   |   |   IF (nwords > max_cfar_words) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "ERROR insufficient memory for CFAR (di, nci)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   ENDLOOP
    |   ENDLOOP
    ENDIF
<---return the return_code
*/
)

/* end of cval_run_cfar.c */
```

```
/*%%=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_run_data_space.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date         Author   Change made
---   -----------  ------   ------------------------------------------------
n/a   15-JUN-1993  fcc      Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_run_data_space

PURPOSE:        This function checks the memory space requirements.


 INPUT                                              SOURCE
 -----                                              ------
 run_def                                            calling routine
 sat_def                                            calling routine
 num_msg                                            calling routine
 cur_log_entry                                      calling routine
 t_cval_run_num_channels                            CVAL global data
 t_cval_run_num_nulled                              CVAL global data
 t_cval_run_num_blanked                             CVAL global data
 t_cval_run_num_dop                                 CVAL global data
 t_cval_run_num_mono                                CVAL global data
 t_cval_run_num_blankers                            CVAL global data
 t_cval_run_M                                       CVAL global data
 t_cval_run_avg_num_snaps_per_upd                   CVAL global data
 t_cval_run_num_weight_vectors[di][nci]             CVAL global data
 t_cval_run_num_pulses_lost                         CVAL global data
 t_cval_run_monopulse_enable                        CVAL global data

 OUTPUT                                             DESTINATION
 ------                                             -----------
 num_msg                                            calling routine
 cur_log_entry                                      calling routine
 return_code                                        calling routine

```

```
    CREATION DATE:  June 15 1993

======================================================================*/
#include "cval.h"

#define NUM_NULLING_CHAN_PER_PH_CTR run_def->ECCM_config.pc_ctr_nulling[syspc].num_nulling_chan
#define PH_CTR_NULLING_CHAN         run_def->ECCM_config.pc_ctr_nulling[syspc].nulling_chan[ch]
#define NEXT_PH_CTR_NULLING_CHAN    run_def->ECCM_config.pc_ctr_nulling[syspc].nulling_chan[dh]
#define CAPTURE_DI_ENABLE           run_def->DI.DI_val[di].capture_DI_enable
#define CAPTURE_NCI_ENABLE          run_def->DI.DI_val[di].NCI.NCI_val[nci].capture_NCI_enable
#define CAPTURE_CPI_ENABLE          run_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_val[cpi].capture_
                                                                                            CPI_enable
#define SEGMENT_DEC_ENABLE          run_def->sig_reduct.segment_dec_enable
#define EXP_PULSE_DUR               run_def->DI.DI_val[di].waveform.exp_pulse_dur
#define DOPPLER_DECIM_STOP          run_def->sig_reduct.doppler_decim.stop
#define DOPPLER_DECIM_START         run_def->sig_reduct.doppler_decim.start
#define DOPPLER_DECIM_STEP          run_def->sig_reduct.doppler_decim.step
#define AUG_DOPPLER_DECIM_STOP      run_def->sig_reduct.aug_doppler_decim.stop
#define AUG_DOPPLER_DECIM_START     run_def->sig_reduct.aug_doppler_decim.start
#define AUG_DOPPLER_DECIM_STEP      run_def->sig_reduct.aug_doppler_decim.step
#define RANGE_DECIM_STOP            run_def->sig_reduct.range_decim.stop
#define RANGE_DECIM_START           run_def->sig_reduct.range_decim.start
#define RANGE_DECIM_STEP            run_def->sig_reduct.range_decim.step
#define TIME_DECIM_STOP             run_def->sig_reduct.time_decim.stop
#define TIME_DECIM_START            run_def->sig_reduct.time_decim.start
#define TIME_DECIM_STEP             run_def->sig_reduct.time_decim.step
#define NUM_PULSES_PROC             run_def->DI.DI_val[di].NCI.NCI_val[nci].num_pulses_proc
#define CAPT_PD_WINDOW              run_def->IR.capt_PD_window
#define CAPT_TARGET_DEWEIGHT        run_def->IR.capt_target_deweight
#define CAPT_DPCA_WEIGHTS           run_def->IR.capt_DPCA_weights
#define NUM_UPDATE_INTS             run_def->ECCM_proc.num_update_ints
#define NUM_DPCA_CELLS              run_def->DI.DI_val[di].ASAR_DPCA.num_DPCA_cells
#define NUM_CC_RNG_GATES            run_def->DI.DI_val[di].ASAR_DPCA.num_CC_RNG_GATES
#define DOPPLER_AUG_SELECT          run_def->DI.DI_val[di].PD.doppler_aug_sel

f_cval_run_data_space(
    t_csid_run_def *run_def,
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
    initialize the return_code to c_spd_success
    pre_dop_bytes = 0
    post_dop_bytes = 0
    post_PC_bytes = 0
    int_prod = 0
    IF (sat_def->data_reduction.magnitude_sqrd == c_spd_true) THEN
    |   psize = sizeof(t_spd_real)
    ELSE
    |   psize = 2 * sizeof(t_spd_real)
    ENDIF
    FOR  = 0 to t_cval_run_num_channels - 1
    LOOP
    |   IF run_def->IR.capt_ESP_input[ch] == c_spd_true THEN
|   |   pre_dop_bytes += 2 * sizeof(t_spd_short)
|   ENDIF
|
|   IF run_def->IR.capt_post_SC[ch] == c_spd_true THEN
|   |   pre_dop_bytes += 2 * sizeof(t_spd_short)
|   ENDIF
ENDLOOP
FOR ch = 0 TO t_cval_run_num_nulled - 1
LOOP
|   IF run_def->IR.capt_nulled[ch] == c_spd_true THEN
|   |   pre_dop_bytes +=  psize
|   ENDIF
ENDLOOP
FOR ch = 0 TO t_cval_run_num_blanked - 1
LOOP
|   IF run_def->IR.capt_ESP_blanked[ch] == c_spd_true THEN
|   |   pre_dop_bytes +=  psize
|   ENDIF
ENDLOOP
IF (t_cval_run_DPCA_mode == c_spd_MASR_DPCA) THEN
|   FOR ch = 0 to t_cval_run_num_mono - 1
|   LOOP
|   |   IF run_def->IR.capt_post_MASR[ch] == c_spd_true THEN
|   |   |   pre_dop_bytes +=  psize
|   |   ENDIF
|   ENDLOOP
ENDIF
FOR ch = 0 TO t_cval_run_num_dop - 1
LOOP
|   IF run_def->IR.capt_post_FA[ch] == c_spd_true THEN
|   |   post_dop_bytes += psize
|   ENDIF
|   IF run_def->IR.capt_post_PD[ch] == c_spd_true THEN
|   |   post_dop_bytes += psize
|   ENDIF
|   IF run_def->IR.capt_RSP_input[ch] == c_spd_true THEN
|   |   post_dop_bytes += psize
|   ENDIF
|   IF run_def->IR.capt_post_PC[ch] == c_spd_true THEN
|   |   post_PC_bytes += psize
|   ENDIF
ENDLOOP
IF (t_cval_run_DPCA_mode == c_spd_ASAR_DPCA) THEN
|   FOR ch = 0 to t_cval_run_num_mono - 1
|   LOOP
|   |   IF run_def->IR.capt_post_CC[ch] == c_spd_true THEN
|   |   |   post_PC_bytes += psize
|   |   ENDIF
|   ENDLOOP
ENDIF
IF t_cval_run_monopulse_enable == c_spd_true &&\
run_def->IR.capt_monopulse_angle == c_spd_true THEN
|   post_PC_bytes += sizeof(t_spd_real)
ENDIF
IF run_def->IR.capt_RCS == c_spd_true THEN
|   post_PC_bytes +=  sizeof(t_spd_real)
```

```
ENDIF
IF run_def->IR.capt_RSP_blanked == c_spd_true THEN
|   post_PC_bytes += sizeof(t_spd_real)
ENDIF

IF run_def->IR.capt_post_NCI == c_spd_true THEN
|   post_NCI_bytes = sizeof(t_spd_real)
ENDIF
norm_bytes = 0
avg_snap_words = 0
num_sys = t_cval_run_num_mono + t_cval_run_num_blankers
FOR sys = 0 to num_sys - 1
LOOP
|   IF run_def->IR.capt_weight_vectors[sys] == c_spd_true THEN
|   |   norm_bytes +=  t_cval_run_M[sys] * 2 * sizeof(t_spd_real) // complex real
|   ENDIF
|   IF run_def->IR.capt_R_r[sys] == c_spd_true THEN
|   |   norm_bytes += t_cval_run_M[sys] * t_cval_run_M[sys] * 2 * sizeof(t_spd_double)
|   ENDIF
|   IF run_def->IR.capt_U_z[sys] == c_spd_true THEN
|   |   norm_bytes += t_cval_run_M[sys] * t_cval_run_M[sys] * 2 * sizeof(t_spd_double)
|   ENDIF
|   IF run_def->IR.capt_snapshots[sys]  == c_spd_true THEN
|   |   avg_snap_words += t_cval_run_M[sys] * t_cval_run_avg_num_snaps_per_upd
|   ENDIF
ENDLOOP
FOR di = 0 TO run_def->DI.DI_len - 1
LOOP
|   IF (CAPTURE_DI_ENABLE == c_spd_true) || (SEGMENT_DEC_ENABLE == c_spd_false) THEN
|   |   int_prod += run_def->IR.capt_pc_replicas *\
|   |   round(EXP_PULSE_DUR * RXFE_SAMPLING_RATE) * 2 * sizeof(t_spd_real) // complex real
|   ENDIF
|   stop = min(DOPPLER_DECIM_STOP, t_cval_run_nbins[di])
|   IF (stop >= DOPPLER_DECIM_START) THEN
|   |   num_dec_bins = floor((stop - DOPPLER_DECIM_START) / DOPPLER_DECIM_STEP) + 1
|   ELSE
|   |   num_dec_bins = 0
|   ENDIF
|   IF DOPPLER_AUG_SELECT == c_spd_none_augment THEN
|   |   stop = min(AUG_DOPPLER_DECIM_STOP, t_cval_run_naug_bins[di])
|   |   IF stop >= AUG_DOPPLER_DECIM_START THEN
|   |   |   num_dec_NCI_bins = floor((stop-AUG_DOPPLER_DECIM_START)/AUG_DOPPLER_DECIM_STEP) +1
|   |   ELSE
|   |   |   num_dec_NCI_bins = 0
|   |   ENDIF
|   ELSE
|   |   num_dec_NCI_bins = num_dec_bins
|   ENDIF
|   IF DOPPLER_AUG_SELECT == c_spd_PC_augment THEN
|   |   num_dec_PC_bins = num_dec_NCI_bins
|   ELSE
|   |   num_dec_PC_bins = num_dec_bins
|   ENDIF
|
|   num_cpis_en_p_di = 0
|   FOR nci = 0 TO run_def->DI.DI_val[di].NCI.NCI_len - 1
|   LOOP
|   |   stop = min(RANGE_DECIM_STOP, t_cval_run_samps_pri[di][nci])
|   |   IF (stop >= RANGE_DECIM_START) THEN
|   |   |   num_dec_ranges = floor((stop - RANGE_DECIM_START) / RANGE_DECIM_STEP) + 1
|   |   ELSE
|   |   |   num_dec_ranges = 0
|   |   ENDIF
|   |   stop = min(TIME_DECIM_STOP, NUM_PULSES_PROC)
|   |   IF (stop >= TIME_DECIM_START) THEN
|   |   |   num_dec_pulses = floor((stop - TIME_DECIM_START) / TIME_DECIM_STEP) + 1
|   |   ELSE
|   |   |   num_dec_pulses = 0
|   |   ENDIF
|   |   IF ((CAPTURE_NCI_ENABLE == c_spd_true) || (SEG_DEC_ENABLE == c_spd_false)) THEN
|   |   |   IF (((num_dec_NCI_bins == 0)||(num_dec_ranges == 0))&& (post_NCI_bytes != 0)) THEN
|   |   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   |   "WARNING no post-NCI data captured (di,nci)"
|   |   |   |   IF there was an error writing the message THEN
|   |   |   |   |   set the return_code to c_spd_failure
|   |   |   |   ENDIF
|   |   |   ENDIF
|   |   |   int_prod += CAPT_PD_WINDOW * (NUM_PULSES_PROC - t_cval_run_num_pulses_lost) * \
|   |   |   sizeof(t_spd_real)
|   |   ENDIF
|   |   num_cpi_enab_per_nci = 0
|   |   FOR cpi = 0 TO run_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_len - 1
|   |   LOOP
|   |   |   IF ((CAPTURE_CPI_ENABLE == c_spd_true) || (SEG_DEC_ENABLE == c_spd_false)) THEN
|   |   |   |   num_cpi_enab_per_nci++
|   |   |   ENDIF
|   |   ENDLOOP
|   |   IF (num_cpi_enab_per_nci > 0) THEN
|   |   |   IF (((num_dec_pulses == 0) || (num_dec_ranges == 0)) && (pre_dop_bytes != 0)) THEN
|   |   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   |   "WARNING no pre-Doppler data captured (di,nci)"
|   |   |   |   IF there was an error writing the message THEN
|   |   |   |   |   set the return_code to c_spd_failure
|   |   |   |   ENDIF
|   |   |   ENDIF
|   |   |   IF (((num_dec_bins == 0) || (num_dec_ranges == 0)) && (post_dop_bytes != 0)) THEN
|   |   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   |   "WARNING no post-Doppler data captured (di,nci)"
|   |   |   |   IF there was an error writing the message THEN
|   |   |   |   |   set the return_code to c_spd_failure
|   |   |   |   ENDIF
|   |   |   ENDIF
|   |   |   IF (((num_dec_PC_bins == 0)||(num_dec_ranges == 0))&& (post_PC_bytes != 0)) THEN
|   |   |   |   Use f_cval_log_msg with index to message indicating\
|   |   |   |   "WARNING no post-PC data captured (di,nci)"
|   |   |   |   IF there was an error writing the message THEN
|   |   |   |   |   set the return_code to c_spd_failure
|   |   |   |   ENDIF
|   |   |   ENDIF
|   |   ENDIF
|   |   int_prod += num_dec_pulses * num_dec_ranges * pre_dop_bytes * num_cpi_enab_per_nci
|   |   int_prod += num_dec_bins * num_dec_ranges * post_dop_bytes * num_cpi_enab_per_nci
```

```
    |   |   int_prod += num_dec_PC_bins * num_dec_ranges * post_PC_bytes  * num_cpi_enab_per_nci
    |   |   int_prod += num_dec_NCI_bins * num_dec_ranges * post_NCI_bytes * num_cpi_enab_per_nci
    |   |
    |   |   int_prod += ceil((t_cval_run_num_weight_vectors[di][nci] + NUM_UPDATE_INTS - 1) *\
    |   |   avg_snap_words) * num_cpi_enab_per_nci * 2 * sizeof(t_spd_real)
    |   |   int_prod += t_cval_run_num_weight_vectors[di][nci] * num_cpi_enab_per_nci * norm_bytes
    |   |   num_cpis_en_p_di += num_cpi_enab_per_nci
    |   ENDLOOP
    |   IF t_cval_run_DPCA_mode == c_spd_ASAR_DPCA THEN
    |   |   int_prod += CAPT_DPCA_WEIGHTS * NUM_DPCA_CELLS * t_cval_run_num_mono *\
    |   |   num_cpis_en_p_di * 2 * sizeof(t_spd_real)
    |   |   int_prod += CAPT_TARGET_DEWEIGHT * NUM_DPCA_CELLS * NUM_CC_RNG_GATES *\
    |   |   t_cval_run_num_mono * num_cpis_en_p_di * sizeof(t_spd_real)
    |   ENDIF
    ENDLOOP
    Use f_cval_log_msg with index to message indicating\
    "INFO total intermediate data is (int_prod) bytes"
    IF there was an error writing the message THEN
    |   set the return_code to c_spd_failure
    ENDIF
<---return the return_code
*/
}

/* end of cval_run_data_space.c */
```

```
/*%%=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      f_cval_sat_def.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---  -----------  ------  ---------------------------------------------
n/a  04-JUN-1993  fcc      Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_def

PURPOSE:        This task performs the validation of the Simulation Run data and the
Satellite and Radar data.

                                                          SOURCE
INPUT                                                     ------
-----                                                     calling routine
sat_def
                                                          DESTINATION
OUTPUT                                                    -----------
------                                                    calling routine
num_msg                                                   calling routine
val_log

CREATION DATE:  June 04 1993

=========================================================================*/
#include "cval.h"

/* Satellite and Radar Data Validation Manager Function */
/* (t_crad_sat_def *) sat_def */
/* (t_spd_int32 *)    num_msg */
/* (char **)          val_log */

f_cval_sat_def(
     t_crad_sat_def *sat_def,
     t_spd_int32    *num_msg,
     char           **val_log
```

D-2 A

```
)
{
/*|
    allocate enough string pointers for each possible message in the val_log
    set val_log to the start of the array of pointers
    initialize cur_log_entry to the start of the val_log array of pointers
    zero the number of error messages
    Use f_cval_sat_ssp(sat_def, num_msg, cur_log_entry) to check SSP
    Use f_cval_sat_freq_range(sat_def, num_msg, cur_log_entry) to check the\
    range of the input frequencies
    Use f_cval_sat_cfa_grid(sat_def, num_msg, cur_log_entry) to check the \
    CFA grid spacings
    Use f_cval_sat_rra_param(sat_def, num_msg, cur_log_entry) to check the\
    RRA antenna parameters
    Use f_cval_sat_pulse(sat_def, num_msg, cur_log_entry) to check the bandwidth,\
    sampling rate, pulse width and compressed width
    Use f_cval_sat_beam(sat_def, num_msg, cur_log_entry) to check that the beam\
    specifications are valid
    Use f_cval_sat_decimation(sat_def, num_msg, cur_log_entry) to check the time\
    and range decimation factors
    Use f_cval_sat_data_space(sat_def, num_msg, cur_log_entry) to check that there\
    is enough system memory
    Use f_cval_sat_tap_delays(sat_def, num_msg, cur_log_entry) to check the tap delays
    Use f_cval_sat_aux_pos(sat_def, num_msg, cur_log_entry) to check the auxiliary positions
    Use f_cval_sat_pri(sat_def, num_msg, cur_log_entry) to check the mutually prime PRIs
    Use f_cval_sat_pc_pos(sat_def, num_msg, cur_log_entry) to check the phase centre positions
    Use f_cval_sat_overlap(sat_def, num_msg, cur_log_entry) to check that the transmissions won't
                                                                                         overlap
    Use f_cval_sat_gen_opt(sat_def, num_msg, cur_log_entry) to check the generation options
    free memory for any of the unused message pointers
*/
}

/* end of cval_sat_def.c */
```

```
#define NEXT_PH_CTR_Z   sat_def->antenna_model.aux_antenna[b].ph_ctr_loc.z
#define DELTA_X         ((PH_CTR_X) - (NEXT_PH_CTR_X))
#define DELTA_Y         ((PH_CTR_Y) - (NEXT_PH_CTR_Y))
#define DELTA_Z         ((PH_CTR_Z) - (NEXT_PH_CTR_Z))
#define GRID_SPACING_X  sat_def->antenna_model.aux_antenna[aux].reg_arr.grid_spacing.x
#define GRID_SPACING_Y  sat_def->antenna_model.aux_antenna[aux].reg_arr.grid_spacing.y

f_cval_sat_aux_pos(
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry,
    t_spd_real     *aperture_time
)
{
/*|
    initialize the return_code to c_spd_success
    FOR aux = 0 to sat_def->antenna_model.aux_count - 1
    LOOP
    |   distance = sqrt((PH_CTR_X ^ 2) + (PH_CTR_Y ^ 2) + (PH_CTR_Z ^ 2))
    |   IF distance > aperture_size THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "WARNING auxiliaries far outside aperture (aux)"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   FOR b = aux + 1 TO sat_def->antenna_model.aux_count - 1
    |   LOOP
    |   |   distance = sqrt((DELTA_X ^ 2) + (DELTA_Y ^ 2) + (DELTA_Z ^ 2))
    |   |   IF (distance < t_cval_lamda_min) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "WARNING auxiliaries too close (aux,b)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   ENDLOOP
    |   IF sat_def->antenna_model.aux_antenna[aux].aux_type == c_spd_regular_array THEN
    |   |   IF (GRID_SPACING_X < 0.25 * c_cval_lambda_min) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "WARNING unrealistic auxiliary x grid-spacing (aux)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF (GRID_SPACING_X > c_cval_lambda_max) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "WARNING grating lobes from high auxiliary x grid-spacing (aux)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF (GRID_SPACING_Y < 0.25 * c_cval_lambda_min) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "WARNING unrealistic auxiliary y grid-spacing (aux)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |  set the return_code to c_spd_failure
    |   |   |  ENDIF
    |   |  ENDIF
    |   |  IF (GRID_SPACING_Y > c_cval_lambda_max) THEN
    |   |   |  Use f_cval_log_msg with index to message indicating\
    |   |   |  "WARNING grating lobes from high auxiliary y grid-spacing (aux)"
    |   |   |  IF there was an error writing the message THEN
    |   |   |   |  set the return_code to c_spd_failure
    |   |   |  ENDIF
    |   |  ENDIF
    |  ENDIF
   ENDLOOP
<---return the return_code
*/
}

/* end of cval_sat_aux_pos.c */
```

```
/*%%===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_sat_beam.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---  -----------  ------  ------------------------------------------------
n/a  08-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_beam

PURPOSE:        This function validates the beam parameters of the Satellite and Radar data.


INPUT                                             SOURCE
-----                                             ------
sat_def                                           calling routine
num_msg                                           calling routine
cur_log_entry                                     calling routine
t_cval_PRI                                        CVAL global data
t_cval_RRA_scan_limit                             CVAL global data
t_cval_CFA_scan_limit                             CVAL global data

OUTPUT                                            DESTINATION
------                                            -----------
return_code                                       calling routine
num_msg                                           calling routine
cur_log_entry                                     calling routine

CREATION DATE:  June 08 1993

==========================================================================*/
#include "cval.h"
#define CFA_RSS_GEN     sat_def->antenna_model.main_antenna.CFA.pattern_RSS_gen
#define PULSE_PER_CPI   sat_def->DI.DI_val[di].NCI.NCI_val[nci].pulse_per_CPI
#define POST_CPI_DELAY  sat_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_val[cpi].post_CPI_delay

f_cval_sat_beam(
```

D-3

```
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
    initialize the return_code to c_spd_success
    set TOD to sat_def->orbital_pos.initial_TOD
    FOR di = 0 TO sat_def->DI.DI_len - 1
    LOOP
    |   set SSP_current to f_sgen_orbit_pos() // Compute the Orbital position
    |   set beam_loc = f_sgen_beam_pointing_conversion_utility()
    |   IF beam_loc.error == c_spd_true THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "ERROR beam centre not on earth 'di'"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ELSEIF sat_def->orbital_pos.loc_method != c_spd_direct_loc_method THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "INFO beam location is 'beam_loc.lat, beam_loc.long' for dwell'di'"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   FOR nci = 0 TO sat_def->DI.DI_val[di].NCI.NCI_len - 1
    |   LOOP
    |   |   FOR cpi = 0 TO sat_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_len - 1
    |   |   LOOP
    |   |   |   beam_in_FOV_flag = f_cval_sat_beam_point_val_tst(sat_def, di, &azim_elev)
    |   |   |   IF beam_in_FOV_flag == c_spd_false THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "ERROR beam position outside field of view 'di,nci,cpi'"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ELSEIF sat_def->concept.concept == c_spd_RRA THEN
    |   |   |   |   axis = f_RSS_Determine_Azimuth_And_Elevation_For_RRA_Axis()
    |   |   |   |   RRA_scan = f_RSS_Determine_RRA_Scan_Angle()
    |   |   |   |   IF RRA_scan > (c_cval_pi / 2) THEN
    |   |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   |   "ERROR beam position behind antenna 'di,nci,cpi'"
    |   |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   |   ENDIF
    |   |   |   |   ENDIF
    |   |   |   |   IF RRA_scan > t_cval_RRA_scan_limit THEN
    |   |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   |   "ERROR beam position cannot be scanned 'di,nci,cpi'"
    |   |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   |   ENDIF
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   |   IF (sat_def->concept.concept) == c_spd_CFA && (CFA_RSS_GEN == c_spd_true) THEN
    |   |   |   |   CFA_scan = f_RSS_Determine_CFA_Scan_Angle()
```

```
      |    |    |    |    IF CFA_scan.w < 0 THEN
      |    |    |    |    |   Use f_cval_log_msg with index to message indicating\
      |    |    |    |    |   "ERROR beam position behind antenna 'di,nci,cpi'"
      |    |    |    |    |   IF there was an error writing the message THEN
      |    |    |    |    |   |   set the return_code to c_spd_failure
      |    |    |    |    |   ENDIF
      |    |    |    |    ENDIF
      |    |    |    |    IF (CFA_scan.u > t_cval_CFA_scan_limit.u) ||\
      |    |    |    |    (CFA_scan.v > t_cval_CFA_scan_limit.v) THEN
      |    |    |    |    |   Use f_cval_log_msg with index to message indicating\
      |    |    |    |    |   "WARNING scanning to beam position causes grating lobes 'di,nci,cpi'"
      |    |    |    |    |   IF there was an error writing the message THEN
      |    |    |    |    |   |   set the return_code to c_spd_failure
      |    |    |    |    |   ENDIF
      |    |    |    |    ENDIF
      |    |    |    ENDIF
      |    |    |    TOD += PULSE_PER_CPI * t_cval_PRI[di][nci] + POST_CPI_DELAY
      |    |    ENDLOOP
      |    ENDLOOP
    ENDLOOP
<---return the return_code
*/
}

/* end of cval_sat_beam.c */
```

```
/*%%=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_sat_beam_tst.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ----------------------------------------------
n/a  21-JUN-1993  fcc      Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_beam_validity_test

PURPOSE:        This function validates the beam pointing longitude and latitude.
It returns a boolean to designate whether the beam is within the Field-Of-View (FOV).


 INPUT                                              SOURCE
 -----                                              ------
 sat_def                                            calling routine
 di                                                 calling routine

 OUTPUT                                             DESTINATION
 ------                                             -----------
 beam_in_FOV_flag                                   calling routine
 azim_elev                                          calling routine

 CREATION DATE:  June 21 1993

 =========================================================================*/

 #include "cval.h"

 #define ALTITUDE      sat_def->orbital_pos.altitude
 #define BEAM_CNTR_LAT sat_def->DI.DI_val[di].location.beam_centre.lat
 #define BEAM_CNTR_LON sat_def->DI.DI_val[di].location.beam_centre.lon
 #define SUBSAT_LAT    sat_def->orbital_pos.SSP_loc.lat
 #define SUBSAT_LON    sat_def->orbital_pos.SSP_loc.lon

 t_spd_boolean f_cval_sat_beam_point_val_tst(
    t_crad_sat_def   *sat_def,
    t_spd_int32      di,
    t_crad_azim_elev *azim_elev
)
{
/*|
    radius_of_earth = 6378000    // metres
    SBR_orbit_radius = radius_of_earth + ALTITUDE   // metres
    SBR_radius_max = sqrt(ALTITUDE * (SBR_orbit_radius + radius_of_earth))
    beam_point_radius = sqrt(radius_of_earth * radius_of_earth +\
    SBR_orbit_radius * SBR_orbit_radius - (2 * radius_of_earth * SBR_radius_max *\
    (cos(BEAM_CNTR_LAT) * cos(SUBSAT_LAT) * cos(BEAM_CNTR_LON - SUBSAT_LON) +\
    sin(BEAM_CNTR_LAT) * sin(SUBSAT_LAT))))
    IF (beam_point_radius <= SBR_radius_max) THEN
    |   set the beam_in_FOV_flag to c_spd_true
    |   set SBR_pos to SbrPosInEcr(SUBSAT_LAT, SUBSAT_LON)
    |   set temp[0] = radius_of_earth
    |   set temp[1] = (c_cval_pi / 2) - BEAM_CNTR_LAT
    |   set temp[2] = BEAM_CNTR_LON
    |   set beam_centre_pos = SpherToCart(temp)
    |   set frd_vect = EcrCartesianToFrdCartesian(beam_centre_pos, SBR_pos)
    |   set vector = CartToSpher(frd_vect)
    |   set azim_elev->azimuth = vector[2]
    |   set azim_elev->elevation = (c_cval_pi / 2) - vector[1]
    ELSE
    |   set the beam_in_FOV_flag to c_spd_false
    ENDIF
<---return the beam_in_FOV_flag
*/
}

/* end of cval_sat_beam_tst.c */
```

D-517

```
/*%%==========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence, 1992

File Name:      cval_sat_cfa_grid.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ----------------------------------------------
n/a   07-JUN-1993 fcc      Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_cfa_grid

PURPOSE:        This function validates the CFA grid spacings of the Satellite and Radar data.


 INPUT                                                  SOURCE
 -----                                                  ------
 t_cval_lambda_min                                      CVAL global data
 t_cval_lambda_max                                      CVAL global data
 sat_def                                                calling routine
 num_msg                                                calling routine
 cur_log_entry                                          calling routine

 OUTPUT                                                 DESTINATION
 ------                                                 -----------
 return_code                                            calling routine
 num_msg                                                calling routine
 cur_log_entry                                          calling routine
 t_cval_CFA_scan_limit                                  CVAL global data

 CREATION DATE:  June 07 1993

 ============================================================================*/

 #include "cval.h"

 #define RSS_GEN        sat_def->antenna_model.main_antenna.CFA.pattern_RSS_gen
 #define GRID_SPACING_X sat_def->antenna_model.main_antenna.CFA.grid_spacing.x
 #define GRID_SPACING_Y sat_def->antenna_model.main_antenna.CFA.grid_spacing.y
```

```
f_cval_sat_cfa_grid(
    t_crad_sat_def    *sat_def,
    t_spd_int32       *num_msg,
    char              **cur_log_entry
)
{
/*|
    set the return code to c_spd_success
    IF (sat_def->concept.concept == c_spd_CFA) && (RSS_GEN == c_spd_true) THEN
    |   // Test grid spacing for x
    |   IF GRID_SPACING_X < 0.25 * t_cval_lambda_min THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "WARNING unrealistic CFA x grid-spacing"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   IF GRID_SPACING_X > t_cval_lambda_min THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "WARNING grating lobes from high CFA x grid-spacing"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   |   t_cval_CFA_scan_limit.u = 1      // not true but disables messages below
    |   ELSEIF GRID_SPACING_X > 0.5 * t_cval_lambda_min THEN
    |   |   t_cval_CFA_scan_limit.u = c_cval_lambda_min / GRID_SPACING_X - 1
    |   ELSE
    |   |   t_cval_CFA_scan_limit.u = 1
    |   ENDIF
    |
    |   // Test grid spacing for y
    |   IF GRID_SPACING_Y < 0.25 * t_cval_lambda_min THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "WARNING unrealistic CFA y grid-spacing"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   IF GRID_SPACING_Y > t_cval_lambda_min THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "WARNING grating lobes from high CFA y grid-spacing"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   |   t_cval_CFA_scan_limit.v = 1      // not true but disables messages below
    |   ELSEIF GRID_SPACING_Y > 0.5 * t_cval_lambda_min THEN
    |   |   t_cval_CFA_scan_limit.v = c_cval_lambda_min / GRID_SPACING_Y - 1
    |   ELSE
    |   |   t_cval_CFA_scan_limit.v = 1
    |   ENDIF
    ENDIF
<---return the return code
*/
}
```

D-5-9

```
/* end of cval_sat_cfa_grid.c */


```

```
/*%%==========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence, 1992

File Name:      cval_sat_data_capt.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR    Date        Author    Change made
---  -----------  ------  ----------------------------------------------
n/a  14-JUN-1993  fcc      Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_data_capt

PURPOSE:        This function adds up the bytes for the input data element.


INPUT                                        SOURCE
-----                                        ------
data_capt_ptr                                calling routine
clut_bytes                                   calling routine
over_sig_bytes                               calling routine
decim_bytes                                  calling routine
jam_bytes                                    calling routine
targ_bytes                                   calling routine
int_prod                                     calling routine
bytes_per_data_element                       calling routine
c_short_size                                 calling routine
c_size                                       calling routine
nsegif                                       calling routine
pattern_size                                 calling routine

OUTPUT                                       DESTINATION
------                                       -----------
clut_bytes                                   calling routine
over_sig_bytes                               calling routine
decim_bytes                                  calling routine
jam_bytes                                    calling routine
targ_bytes                                   calling routine
int_prod                                     calling routine

```

```
CREATION DATE:  June 14 1993

================================================================================*/

#include "cval.h"

f_cval_sat_data_capt(
    t_crad_data_capture *data_capt_ptr,
    t_spd_int32         *clut_bytes,
    t_spd_int32         *over_sig_bytes,
    t_spd_int32         *decim_bytes,
    t_spd_int32         *jam_bytes,
    t_spd_int32         *targ_bytes,
    t_spd_int32         *int_prod,
    t_spd_int32         bytes_per_data_element,
    t_spd_int32         c_short_size,
    t_spd_int32         c_size,
    t_spd_int32         nsegif,
    t_spd_int32         pattern_size
)
{
 /*|
    IF data_capt_ptr->capt_clutter == c_spd_true THEN
    |   clut_bytes += bytes_per_data_element
    ENDIF
    IF data_capt_ptr->capt_ext_noise == c_spd_true THEN
    |   over_sig_bytes += bytes_per_data_element
    ENDIF
    IF data_capt_ptr->capt_composite == c_spd_true THEN
    |   over_sig_bytes += bytes_per_data_element
    ENDIF
    IF data_capt_ptr->capt_post_nl == c_spd_true THEN
    |   over_sig_bytes += bytes_per_data_element
    ENDIF
    IF data_capt_ptr->capt_post_filt == c_spd_true THEN
    |   decim_bytes += bytes_per_data_element
    ENDIF
    IF data_capt_ptr->capt_post_sd == c_spd_true THEN
    |   decim_bytes += bytes_per_data_element
    ENDIF
    IF data_capt_ptr->capt_post_ad == c_spd_true THEN
    |   decim_bytes += c_short_size
    ENDIF
    IF data_capt_ptr->capt_jammer == c_spd_true THEN
    |   jam_bytes += bytes_per_data_element
    ENDIF
    IF data_capt_ptr->capt_target == c_spd_true THEN
    |   targ_bytes += bytes_per_data_element
    ENDIF
    IF data_capt_ptr->capt_filt_rsp == c_spd_true THEN
    |   int_prod += nsegif * c_size
    ENDIF
    IF data_capt_ptr->capt_ant == c_spd_true THEN
    |   int_prod += pattern_size
    ENDIF
*/
}

/* end of cval_data_capt.c */
```

```
/*%%===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence, 1992

File Name:      cval_sat_data_space.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author  Change made
---   ----------- ------  ------------------------------------------------
n/a   07-JUN-1993 fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_data_space

PURPOSE:        This function checks if there is enough memory for the data.


INPUT                                          SOURCE
-----                                          ------
sat_def                                        calling routine
num_msg                                        calling routine
cur_log_entry                                  calling routine
t_cval_samps_pri                               CVAL global data
t_cval_pri                                     CVAL global data

OUTPUT                                         DESTINATION
------                                         -----------
return_code                                    calling routine
num_msg                                        calling routine
cur_log_entry                                  calling routine

CREATION DATE:  June 07 1993

============================================================================*/
#include "cval.h"

#define NUM_DPCA_GRPS          sat_def->antenna_model.main_antenna.CFA.num_DPCA_grps
#define NUM_PH_CTR_PER_GRP     sat_def->antenna_model.main_antenna.CFA.num_ph_ctr_per_grp
#define RANGE_START            sat_def->range_decim.start
#define RANGE_STOP             sat_def->range_decim.stop
#define RANGE_STEP             sat_def->range_decim.step
```

```
#define TIME_START          sat_def->time_decim.start
#define TIME_STOP           sat_def->time_decim.stop
#define TIME_STEP           sat_def->time_decim.step
#define CLUTTER_FACTOR      sat_def->gen_options.clutter_factor
#define JAMMER_NOISE_FACTOR sat_def->gen_options.jammer_noise_factor
#define MOTION_COMP         sat_def->antenna_model.main_rxfe.motion_comp
#define PULSE_PER_CPI       sat_def->DI.DI_val[di].NCI.NCI_val[nci].pulse_per_CPI
#define POST_CPI_DELAY     sat_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_val[cpi].post_CPI_delay
#define CAPTURE_CPI_ENABLE  sat_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_val[cpi].capture_CPI_
                                                                                        enable
#define TAP_DELAY          sat_def->antenna_model.aux_antenna[aux].delay_taps[tap]
#define AUX_COUNT          sat_def->antenna_model.aux_count
#define TAP_COUNT          sat_def->antenna_model.aux_antenna[aux].tap_count

f_cval_sat_data_space(
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
    initialize the return_code to c_spd_success
    c_size = 2 * sizeof(t_spd_real)        // size of a complex real
    f_size = sizeof(t_spd_real)            // size of a real
    c_short_size = 2 * sizeof(t_spd_short) // size of a complex short
    IF sat_def->concept.concept == c_spd_CFA THEN
    |   num_ph_ctrs = NUM_DPCA_GRPS * NUM_PH_CTR_PER_GRP
    ELSE
    |   num_ph_ctrs = 1
    ENDIF
    ndoft = 0
    FOR aux = 0 TO AUX_COUNT - 1
    LOOP
    |   FOR tap = 0 TO TAP_DELAY
    |   LOOP
    |   |   ndoft++
    |   ENDLOOP
    ENDLOOP
    num_channels = num_ph_ctrs + ndoft
    bytes_per_samp = num_channels * c_size
    clear clut_bytes
    clear over_sig_bytes
    clear decim_bytes
    clear jam_bytes
    clear targ_bytes
    clear total_data
    clear int_prod
    IF sat_def->data_reduction.magnitude_sqrd == c_spd_true THEN
    |   bytes_per_data_element = f_size   // size of float
    ELSE
    |   bytes_per_data_element = c_size   // size of complex float
    ENDIF
    nsegif = 1024                              // RSS IF filter size
    IF sat_def->concept.concept == c_spd_RRA THEN
    |   nu = 16384                             // RSS RRA pattern length
    |   main_pat_size = nu * 2.5 * c_size * 2       // 2.5 patterns, 8 bytes per sample, tx and rx
```

```
ELSE
|   nu = 2048                          // RSS CFA pattern length
|   main_pat_size = nu * 4 * c_size * 2      // 4 patterns, 8 bytes per sample, tx and rx
ENDIF
nu = 2048
aux_pat_size = nu * 4 * c_size             // 4 patterns, rx only

// Calculate the data_capture data space required for the auxiliary antennas
FOR aux = 0 TO AUX_COUNT - 1
LOOP
|   set data_capt_ptr to the address of sat_def->antenna_model.aux_antenna[aux].data_capture
|   set the pattern_size to the aux_pat_size
|   Use f_cval_sat_data_capt() to add up the data capture elements for data_capt_ptr
|   FOR tap = 0 to TAP_COUNT - 1
|   LOOP
|   |   set data_capt_ptr to address of\
|   |   sat_def->antenna_model.aux_antenna[aux].delay_taps[tap].data_capture
|   |   Use f_cval_sat_data_capt() to add up the data capture elements for data_capt_ptr
|   ENDLOOP
ENDLOOP
// Calculate the data_capture data space required for the main antenna
IF sat_def->concept.concept == c_spd_RRA THEN
|   set data_capt_ptr to the address of sat_def->antenna_model.main_antenna.RRA.data_capture
|   set the pattern_size to the main_pat_size
|   Use f_cval_sat_data_capt() to add up the data capture elements for data_capt_ptr
ELSE
|   set data_capt_ptr to the address of sat_def->antenna_model.main_antenna.CFA.data_capture
|   set the pattern_size to the main_pat_size
|   Use f_cval_sat_data_capt() to add up the data capture elements for data_capt_ptr
|   FOR grp = 0 TO NUM_DPCA_GRPS - 1
|   LOOP
|   |   set data_capt_ptr to the address of\
|   |   sat_def->antenna_model.main_antenna.CFA.ph_ctr_dat_capt[grp]
|   |   set the pattern size to main_pat_size
|   |   Use f_cval_sat_data_capt() to add up the data capture elements for data_capt_ptr
|   ENDLOOP
ENDIF

rstep_c = max(round(RANGE_STEP * CLUTTER_FACTOR) / CLUTTER_FACTOR, 1)
IF (abs(rstep_c - RANGE_STEP) > 1.0E-6 * rstep_c) THEN
|   Use f_cval_log_msg with index to message indicating\
|   "INFO range step rounded to (rstep_c) for clutter samples"
|   IF there was an error writing the message THEN
|   |   set the return_code to c_spd_failure
|   ENDIF
ENDIF
rstep_o = max(round(RANGE_STEP * JAMMER_NOISE_FACTOR) / JAMMER_NOISE_FACTOR, 1)
IF (abs(rstep_o - RANGE_STEP) > 1.0E-6 * rstep_o) THEN
|   Use f_cval_log_msg with index to message indicating\
|   "INFO range step rounded to (rstep_o) for pre-I F samples"
|   IF there was an error writing the message THEN
|   |   set the return_code to c_spd_failure
|   ENDIF
ENDIF
rstep_s = max(round(RANGE_STEP.step), 1)
IF (abs(rstep_s - RANGE_STEP) > 1.0E-6 * rstep_s) THEN
```

```
    |   Use f_cval_log_msg with index to message indicating\
    |   "INFO range step rounded to (rstep_s) for post-I F samples"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
    set TOD to sat_def->orbital_pos.initial_TOD
    FOR di = 0 TO sat_def->DI.DI_len - 1
    LOOP
    |   FOR nci = 0 TO sat_def->DI.DI_val[di].NCI.NCI_len - 1
    |   LOOP
    |   |   FOR cpi = 0 TO sat_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_len - 1
    |   |   LOOP
    |   |   |   IF (MOTION_COMP == c_spd_false (off)) THEN
    |   |   |   |   extra_samps = f_RSS_Determine_Number_Of_Extra_Samples()
    |   |   |   ELSE
    |   |   |   |   extra_samps = 0
    |   |   |   ENDIF
    |   |   |   cpi_data = (PULSE_PER_CPI * t_cval_samps_pri[di][nci]+extra_samps)*bytes_per_samp
    |   |   |   IF (cpi_data > c_cval_max_cpi) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "ERROR too much signal data (CPI) (di,nci.cpi)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ELSEIF (cpi_data > c_cval_comfort_cpi) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "WARNING nearly too much signal data (CPI), limited\
    |   |   |   |   nulling updates(di,nci.cpi)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   |   total_data += cpi_data
    |   |   |   IF (total_data > c_cval_max_data) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "ERROR too much signal data (di,nci.cpi)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   |   TOD += PULSE_PER_CPI * 1.0E-6 * t_cval_pri[di][nci] + 1.0E-3 * POST_CPI_DELAY
    |   |   ENDLOOP
    |   |   stop = min(RANGE_STOP, t_cval_samps_pri[di][nci])
    |   |   IF stop >= RANGE_START THEN
    |   |   |   num_dec_clutter_ranges = floor((stop - RANGE_START) / rstep_c) + 1
    |   |   |   num_dec_over_ranges = floor((stop - RANGE_START) / rstep_o) + 1
    |   |   |   num_dec_ranges = floor((stop - RANGE_START) / rstep_s) + 1
    |   |   ELSE
    |   |   |   clear num_dec_clutter_ranges
    |   |   |   clear num_dec_over_ranges
    |   |   |   clear num_dec_ranges
    |   |   ENDIF
    |   |   stop = min(TIME_STOP, PULSE_PER_CPI)
    |   |   IF (stop >= TIME_START) THEN
    |   |   |   num_dec_pulses = floor((stop - TIME_START) / TIME_STEP) + 1
```

D-525

```
    |   |   ELSE
    |   |   |   num_dec_pulses = 0
    |   |   ENDIF
    |   |   num_cpi_enamble_per_nci = 0
    |   |   FOR cpi = 0 TO sat_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_len - 1
    |   |   LOOP
    |   |   |   IF CAPTURE_CPI_ENABLE == c_spd_true || seg_dec_enable == 'no' THEN
    |   |   |   |   num_cpi_enamble_per_nci++
    |   |   |   ENDIF
    |   |   ENDLOOP
    |   |   int_prod += num_dec_pulses * num_dec_clutter_ranges *\
    |   |   clut_bytes * num_cpi_enamble_per_nci
    |   |   int_prod += num_dec_pulses * num_dec_over_ranges *\
    |   |   over_sig_bytes * num_cpi_enamble_per_nci
    |   |   int_prod += num_dec_pulses * num_dec_ranges * decim_bytes * num_cpi_enamble_per_nci
    |   |   jam_prod += num_dec_pulses * num_dec_over_ranges * jam_bytes * num_cpi_enamble_per_nci
    |   |   targ_prod += num_dec_pulses * num_dec_over_ranges * targ_bytes *\
    |   |   num_cpi_enamble_per_nci
    |   |   IF num_cpi_enamble_per_nci > 0 THEN
    |   |   |   IF (((num_dec_pulses == 0) || (num_dec_ranges == 0)) && (decim_bytes > 0)) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "WARNING no intermediate post-I F signal data captured (di,nci)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   |   IF (((num_dec_pulses == 0) || (num_dec_over_ranges == 0)) &&\
    |   |   |   (over_sig_bytes + jam_bytes + targ_bytes > 0)) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "WARNING no intermediate pre-I F signal data captured (di,nci)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   |   IF (((num_dec_pulses == 0) || (num_dec_clutter_ranges == 0)) &&\
    |   |   |   (clut_bytes > 0)) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "WARNING no intermediate clutter data captured (di,nci)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   ENDIF
    |   ENDLOOP
    ENDLOOP
    Use f_cval_log_msg with index to message indicating\
    "INFO total intermediate data is (int_prod) bytes +\
    (jam_prod) bytes per captured jammer + (targ_prod) bytes per captured target)"
    IF there was an error writing the message THEN
    |   set the return_code to c_spd_failure
    ENDIF
<---return the return_code
*/
}

/* end of cval_sat_data_space.c */
```

D-517

D-529

```
/*%%=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_sat_decimation.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---  -----------  ------  --------------------------------------------------
n/a  07-JUN-1993  fcc      Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_decimation

PURPOSE:        This function checks the range decimation.


INPUT                                               SOURCE
-----                                               ------
sat_def                                             calling routine
num_msg                                             calling routine
cur_log_entry                                       calling routine

OUTPUT                                              DESTINATION
------                                              -----------
return_code                                         calling routine
num_msg                                             calling routine
cur_log_entry                                       calling routine

CREATION DATE:  June 07 1993

==========================================================================*/
#include "cval.h"

#define RANGE_START sat_def_ptr->data_reduction.range_decim.start
#define RANGE_STOP  sat_def_ptr->data_reduction.range_decim.stop
#define TIME_START  sat_def_ptr->data_reduction.time_decim.start
#define TIME_STOP   sat_def_ptr->data_reduction.time_decim.stop

f_cval_sat_decimation(
    t_crad_sat_def *sat_def,
```

D-52-

```
       t_spd_int32    *num_msg,
       char           **cur_log_entry
)
{
/*|
       initialize the return_code to c_spd_success
       IF RANGE_START > RANGE_STOP THEN
       |   Use f_cval_log_msg with index to message indicating\
       |   "WARNING no intermediate products: range start > stop"
       |   IF there was an error writing the message THEN
       |   |   set the return_code to c_spd_failure
       |   ENDIF
       ENDIF
       initialize the return_code to c_spd_success
       IF TIME_START > TIME_STOP THEN
       |   Use f_cval_log_msg with index to message indicating\
       |   "WARNING no intermediate products: time start > stop"
       |   IF there was an error writing the message THEN
       |   |   set the return_code to c_spd_failure
       |   ENDIF
       ENDIF
<---return the return_code
*/
}

/* end of cval_sat_decimation.c */
```

```
/*%%============================================================================
SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence, 1992

File Name:      cval_sat_freq_range.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---  -----------  ------  --------------------------------------------------
n/a  07-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_freq_range

PURPOSE:        This function validates the frequency range of the Satellite and Radar data.


 INPUT                                                    SOURCE
 -----                                                    ------
 sat_def                                                  calling routine
 num_msg                                                  calling routine
 cur_log_entry                                            calling routine

 OUTPUT                                                   DESTINATION
 ------                                                   -----------
 t_cval_lambda_min                                        CVAL global data
 t_cval_lambda_max                                        CVAL global data
 return_code                                              calling routine
 num_msg                                                  calling routine
 cur_log_entry                                            calling routine


 CREATION DATE:  June 07 1993

 ============================================================================*/
 #include "cval.h"

 #define RF_FREQ sat_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_val[cpi].RF_freq

 f_cval_sat_freq_range(
     t_crad_sat_def *sat_def,
     t_spd_int32    *num_msg,
char            **cur_log_entry

/*|
    set the return code to c_spd_success
    freq_min = sat_def->concept.freq_mid - sat_def->concept.bandwidth/2.0E03
    freq_max = sat_def->concept.freq_mid + sat_def->concept.bandwidth/2.0E03
    IF sat_def->concept.freq_band == c_spd_L_band THEN
    |   IF sat_def->concept.freq_max > c_cval_max_L_band_frequency THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "WARNING maximum frequency outside L-band"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   IF sat_def->concept.freq_min < c_cval_min_L_band_frequency THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "WARNING minimum frequency outside L-band"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    ELSE
    |   IF sat_def->concept.freq_max > c_cval_max_S_band_frequency THEN
    |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   "WARNING maximum frequency outside S-band"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   IF sat_def->concept.freq_min < c_cval_min_S_band_frequency THEN
    |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   "WARNING minimum frequency outside S-band"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    ENDIF
    t_cval_lambda_min = 0.3 / freq_max    // metres
    t_cval_lambda_max = 0.3 / freq_min    // metres

    // Now check the RF frequency parameter
    FOR di = 0 TO sat_def->DI.DI_len - 1
    LOOP
    |   FOR nci = 0 TO sat_def->DI.DI_val[di].NCI.NCI_len - 1
    |   LOOP
    |   |   FOR cpi = 0 TO sat_def->DI.DI_val[di].NCI.NCI_val.nci].CPI.CPI_len - 1
    |   |   LOOP
    |   |   |   IF (RF_FREQ < freq_min) || (RF_FREQ > freq_max) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "WARNING RF frequency out of defined band (di,nci,cpi)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   ENDLOOP
    |   ENDLOOP
    ENDLOOP
<---return the return code
*/
}

/* end of cval_sat_freq_range.c */


```

```
/*%%============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_sat_gen_opt.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date         Author   Change made
---  -----------  ------  ------------------------------------------------
n/a  08-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_gen_opt

PURPOSE:        This function checks the generation options in the satellite and radar data.


INPUT                                                   SOURCE
-----                                                   ------
sat_def                                                 calling routine
num_msg                                                 calling routine
cur_log_entry                                           calling routine

OUTPUT                                                  DESTINATION
------                                                  -----------
return_code                                             calling routine
num_msg                                                 calling routine
cur_log_entry                                           calling routine

CREATION DATE:  June 08 1993

============================================================================*/
#include "cval.h"


f_cval_sat_gen_opt(
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
```

5-735

```
/*|
    initialize the return_code to c_spd_success
    IF sat_def->gen_options.clutter_factor > sat_def->gen_options.jammer_noise_factor THEN
    |   Use f_cval_log_msg with index to message indicating\
    |   "ERROR clutter oversampling factor must not exceed jammer and\
    |   noise oversampling factor"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
<---return the return_code
*/
}

/* end of cval_sat_gen_opt.c */
```

```
/*%%============================================================================
SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_sat_overlap.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ---------------------------------------------
n/a   08-JUN-1993 fcc      Initial Release

CSU_NAME:       CVAL
FUNCTION:       f_cval_sat_overlap

PURPOSE:        This function checks for transmitting while receiving and overlapping dwells.


INPUT                                              SOURCE
-----                                              ------
sat_def                                            calling routine
num_msg                                            calling routine
cur_log_entry                                      calling routine
t_cval_PRI                                         CVAL global data
t_cval_PRF                                         CVAL global data

OUTPUT                                             DESTINATION
------                                             -----------
return_code                                        calling routine
num_msg                                            calling routine
cur_log_entry                                      calling routine

CREATION DATE:  June 08 1993

============================================================================*/
#include "cval.h"

#define PULSE_PER_CPI  sat_def->DI.DI_val[di].NCI.NCI_val[nci].pulse_per_CPI
#define POST_CPI_DELAY sat_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_val[cpi].post_CPI_delay
#define EX_PULSE_DUR   sat_def->DI.DI_val[di].waveform.ex_pulse_dur

f_cval_sat_overlap(
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
    initialize the return_code to c_spd_success
    set TOD to sat_def->orbital_pos.initial_TOD
    ton[0] = 0
    cpi_count = 0
    FOR di = 0 to sat_def->DI.DI_len - 1
    LOOP
    |   boresight_delay[di] = f_RSS_Determine_Boresight_Delay()
    |   IF di > 0 THEN
    |   |   IF (boresight_delay[di] + last_delay < boresight_delay[di-1]) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "ERROR post CPI delay not long enough between dwells (di, di-1)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   ENDIF
    |   FOR nci = 0 to sat_def->DI.DI_val[di].NCI.NCI_len - 1
    |   LOOP
    |   |   FOR cpi = 0 to sat_def->DI.DI_val[di].NCI.NCI_val[nci].CPI.CPI_len - 1
    |   |   LOOP
    |   |   |   cpi_boresight_delay = f_RSS_Determine_Boresight_Delay()
    |   |   |   toff[cpi_count] = ton[cpi_count] + PULSE_PER_CPI * 1.0E-06 * t_cval_PRI[di][nci]
    |   |   |   ron[cpi_count] = ton[cpi_count] + cpi_boresight_delay
    |   |   |   roff[cpi_count] = toff[cpi_count] + cpi_boresight_delay
    |   |   |   ton[cpi_count+1] = toff[cpi_count] + 1.0E-03 * POST_CPI_DELAY
    |   |   |   duty[cpi_count] = 1.0E-03 * EXP_PULSE_DUR * t_cval_PRF[di][nci]
    |   |   |   dw[cpi_count] = di
    |   |   |   nc[cpi_count] = nci
    |   |   |   cp[cpi_count] = cpi
    |   |   |   last_delay = 1.0E-03 * POST_CPI_DELAY
    |   |   |   cpi_count++
    |   |   |   TOD += PULSE_PER_CPI * 1.0E-6 * t_cval_PRI[di][nci] + 1.0E-03 * POST_CPI_DELAY
    |   |   ENDLOOP
    |   ENDLOOP
    ENDLOOP
    t = 0
    FOR r = 0 to cpi_count - 1
    LOOP
    |   WHILE ton[t] < roff[r]
    |   LOOP
    |   |   IF (toff[t] > ron[r] THEN
    |   |   |   frac = (min(toff[t],roff[r]) - max(ton[t], ron[r])) / (roff[r] - ron[r])
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "WARNING dwell (dw[t]) nci (nc[t]) cpi (cp[t]) transmitting while\
    |   |   |   dwell (dw[r]) nci (nc[r]) cpi (cp[r]) receiving,\
    |   |   |   (frac * 100) percent of receive cpi, (duty[t] * 100) percent\
    |   |   |   pulse duty cycle "
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
```

```
      |   |   ENDIF
      |   |   t++
      |   ENDLOOP
    ENDLOOP
<---return the return_code
*/
}

/* end of cval_sat_overlap.c */

```

```
/*%%==========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_sat_pc_pos.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ---------------------------------------------------
n/a   08-JUN-1993 fcc      Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_beam

PURPOSE:        This function validates the phase centre positions of the Satellite and Radar data.


 INPUT                                              SOURCE
 -----                                              ------
 sat_def                                            calling routine
 num_msg                                            calling routine
 cur_log_entry                                      calling routine
 t_cval_PRI                                         CVAL global data

 OUTPUT                                             DESTINATION
 ------                                             -----------
 return_code                                        calling routine
 num_msg                                            calling routine
 cur_log_entry                                      calling routine

 CREATION DATE:  June 08 1993

 ==========================================================================*/
 #include "cval.h"

 #define DISP_X              sat_def->antenna_model.main_antenna.CFA.disp[grp].x
 #define DISP_Y              sat_def->antenna_model.main_antenna.CFA.disp[grp].y
 #define APERTURE_X          sat_def->antenna_model.main_antenna.CFA.aperture.x
 #define APERTURE_Y          sat_def->antenna_model.main_antenna.CFA.aperture.y
 #define NUM_PH_CTR_PER_GRP  sat_def->antenna_model.main_antenna.CFA.num_ph_ctr_per_grp
 #define PRI_MULTIPLIER      sat_def->antenna_model.main_antenna.CFA.PRI_multiplier
```

```
#define NUM_DPCA_GRPS       sat_def->antenna_model.main_antenna.CFA.num_DPCA_grps

f_cval_sat_pc_pos(
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
    initialize the return_code to c_spd_success
    FOR grp = 0 to sat_def->antenna_model.main_antenna.CFA.num_DPCA_grps-1
    LOOP
    |   IF (abs(DISP_X) > APERTURE_X/4) THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "WARNING aperture not used much (x-centre) (grp)"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   IF (abs(DISP_X) > APERTURE_X/2) THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "ERROR phase centre outside aperture (x-centre) (grp)"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   IF (abs(DISP_Y) > APERTURE_Y/4) THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "WARNING aperture not used much (y-centre) (grp)"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   IF (abs(DISP_Y) > APERTURE_Y/2) THEN
    |   |   Use f_cval_log_msg with index to message indicating\
    |   |   "ERROR phase centre outside aperture (y-centre) (grp)"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    ENDLOOP
    IF NUM_PH_CTR_PER_GRP > 1 THEN
    |   orbit_coeff = f_RSS_Compute_Orbital_Model_Coefficients()
    |   xmult = 0.5 * (NUM_PH_CTR_PER_GRP - 1) * PRI_MULTIPLIER * orbit_coeffic.v_sat
    |   FOR di = 0 to sat_def->DI.DI_len - 1
    |   LOOP
    |   |   FOR nci = 0 to sat_def->DI.DI_val[di].NCI.NCI_len - 1
    |   |   LOOP
    |   |   |   dx = xmult * t_cval_PRI[d][n]
    |   |   |   xmax = 0
    |   |   |   FOR grp = 0 to NUM_DPCA_GRPS - 1
    |   |   |   LOOP
    |   |   |   |   xmax = max(xmax, abs(DISP_X + dx), abs(DISP_X - dx))
    |   |   |   ENDLOOP
    |   |   |   IF (xmax > APERTURE_X/4) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "WARNING aperture not used much (x-outer) (di,nci)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   |   IF (xmax > APERTURE_X/2) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "ERROR phase centre outside aperture (x-outer) (di,nci)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   ENDLOOP
    |   ENDLOOP
    ENDIF
<---return the return value
*/
}

/* end of cval_sat_pc_pos.c */


```

```
/*%%=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_sat_pri.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---  -----------  ------  -------------------------------------------------
n/a  08-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_pri

PURPOSE:        This function validates the PRI parameters of the Satellite and Radar data.


INPUT                                               SOURCE
-----                                               ------
sat_def                                             calling routine
num_msg                                             calling routine
cur_log_entry                                       calling routine
t_cval_samps_pri                                    CVAL global data

OUTPUT                                              DESTINATION
------                                              -----------
return_code                                         calling routine
num_msg                                             calling routine
cur_log_entry                                       calling routine

CREATION DATE:  June 08 1993

=========================================================================*/
#include "cval.h"

f_cval_sat_pri(
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
```

D-545

```
/*|
    initialize the return_code to c_spd_success
    FOR di = 0 TO sat_def->DI.DI_len - 1
    LOOP
    |   FOR nci = 0 TO sat_def->DI.DI_val[di].NCI.NCI_len - 1
    |   LOOP
    |   |   FOR m = nci + 1 TO sat_def->DI.DI_val[di].NCI.NCI_len - 1
    |   |   LOOP
    |   |   |   IF (gcd(t_cval_samps_pri[di][nci], t_cval_samps_pri[di][m]) != 1) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "WARNING PRIs not mutually prime integers (di,nci,m)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   ENDLOOP
    |   ENDLOOP
    ENDLOOP
<---return the return_code
*/
}

/* end of cval_sat_pri.c */


```

```
/*%%=============================================================================
SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_sat_pulse.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR    Date        Author    Change made
---  -----------  ------  ------------------------------------------------
n/a  08-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_pulse

PURPOSE:        This function checks the bandwidth, PRF, sampling rate, pulse_width,
compressed pulse width compatibility.


INPUT                                                   SOURCE
-----                                                   ------
sat_def                                                 calling routine
num_msg                                                 calling routine
cur_log_entry                                           calling routine

OUTPUT                                                  DESTINATION
------                                                  -----------
return_code                                             calling routine
num_msg                                                 calling routine
cur_log_entry                                           calling routine
t_cval_samps_PRI                                       CVAL global data
t_cval_PRF                                             CVAL global data
t_cval_PRI                                              CVAL global data

CREATION DATE:  June 08 1993

============================================================================*/
#include "cval.h"

#define RXFE_BANDWIDTH      sat_def->antenna_model.main_rxfe.bandwidth
#define RXFE_SAMPLING_RATE  sat_def->antenna_model.main_rxfe.sampling_rate
#define PULSE_TYPE          sat_def->antenna_model.main_rxfe.pulse_type
#define EX_PULSE_DUR        sat_def->DI.DI_val[di].waveform.ex_pulse_dur
```

```
#define COMPR_PULSE_WIDTH  sat_def->DI.DI_val[di].waveform.compr_pulse_width
#define NCI_PRF            sat_def->DI.DI_val[di].NCI.NCI_val[nci].PRF
#define SAMPLES_PER_PRI    sat_def->DI.DI_val[di].NCI.NCI_val[nci].samples_per_PRI
#define PRF_ENTERED        sat_def->DI.DI_val[di].NCI.NCI_val[nci].PRF_entered

f_cval_sat_pulse(
    t_crad_sat_def  *sat_def,
    t_spd_int32     *num_msg,
    char            **cur_log_entry
)
{
/*|
    set the return code to c_spd_success
    rx_cor = 1.0 / RXFE_BANDWIDTH
    sampling_int = 1.0 / RXFE_SAMPLING_RATE
    IF (RXFE_SAMPLING_RATE > 2 * RXFE_BANDWIDTH ) THEN
     |  Use f_cval_log_msg with index to message indicating\
     |  "WARNING oversampled relative to rx bandwdith"
     |  IF there was an error writing the message THEN
     |  |   set the return_code to c_spd_failure
     |  ENDIF
    ENDIF
    FOR di = 0 TO sat_def->DI.DI_len - 1
    LOOP
     |  IF EX_PULSE_DUR < sampling_int THEN
     |  |  Use f_cval_log_msg with index to message indicating\
     |  |  "WARNING pulse duration too short (di), relative to sampling rate"
     |  |  IF there was an error writing the message THEN
     |  |  |   set the return_code to c_spd_failure
     |  |  ENDIF
     |  ENDIF
     |  IF EX_PULSE_DUR > sampling_int * 512 THEN
     |  |  Use f_cval_log_msg with index to message indicating\
     |  |  "ERROR pulse duration too long (di), more than 512 samples"
     |  |  IF there was an error writing the message THEN
     |  |  |   set the return_code to c_spd_failure
     |  |  ENDIF
     |  ENDIF
     |  IF COMPR_PULSE_WIDTH < sampling_int THEN
     |  |  Use f_cval_log_msg with index to message indicating\
     |  |  "WARNING undersampled relative to compressed pulse duration (di)"
     |  |  IF there was an error writing the message THEN
     |  |  |   set the return_code to c_spd_failure
     |  |  ENDIF
     |  ENDIF
     |  IF COMPR_PULSE_WIDTH < 0.5 * rx_cor THEN
     |  |  Use f_cval_log_msg with index to message indicating\
     |  |  "WARNING receiver bandwidth widens compressed pulse (di)"
     |  |  IF there was an error writing the message THEN
     |  |  |   set the return_code to c_spd_failure
     |  |  ENDIF
     |  ENDIF
     |  IF COMPR_PULSE_WIDTH > EXP_PULSE_DUR THEN
     |  |  Use f_cval_log_msg with index to message indicating\
     |  |  "ERROR compressed pulse duration longer than expanded (di)"
     |  |  IF there was an error writing the message THEN
```

```
      |   |   |   set the return_code to c_spd_failure
      |   |   ENDIF
      |   ENDIF
      |   samples_per_pulse[di] = round(EXP_PULSE_DUR * RXFE_SAMPLING_RATE)
      |   // use a local variable to round the samples per pulse
      |   FOR nci = 0 TO sat_def->DI.DI_val[di].NCI.NCI_len - 1
      |   LOOP
      |   |   IF PRF_ENTERED == c_spd_true (entered directly) THEN
      |   |   |   t_cval_samps_PRI[di][nci] = round(1.0E03 * RXFE_SAMPLING_RATE/NCI_PRF)
      |   |   |   t_cval_PRF[di][nci] = 1.0E03 * RXFE_SAMPLING_RATE/samps[di][nci]
      |   |   |   IF t_cval_PRF[di][nci] - NCI_PRF > 1.0E-06 * t_cval_PRF[di][nci] THEN
      |   |   |   |   Use f_cval_log_msg with index to message indicating\
      |   |   |   |   "INFO PRF rounded to t_cval_PRF[di][nci] 'di,nci'"
      |   |   |   |   IF there was an error writing the message THEN
      |   |   |   |   |   set the return_code to c_spd_failure
      |   |   |   |   ENDIF
      |   |   |   ELSE
      |   |   |   |   t_cval_samps_PRI[di][nci]  = SAMPLES_PER_PRI
      |   |   |   |   t_cval_samps_PRI[di][nci] = samples_per_PRI[di][nci]
      |   |   |   |   t_cval_PRF[di][nci] = 1.0E03 * RXFE_SAMPLING_RATE/t_cval_samps_PRI[di][nci]
      |   |   |   ENDIF
      |   |   ENDIF
      |   |   t_cval_PRI[di][nci] = 1 / t_cval_PRF[di][nci]
      |   |   IF t_cval_PRI[di][nci] < EXP_PULSE_DUR THEN
      |   |   |   Use f_cval_log_msg with index to message indicating\
      |   |   |   "ERROR PRF too high (di,nci), PRI less than pulse duration"
      |   |   |   IF there was an error writing the message THEN
      |   |   |   |   set the return_code to c_spd_failure
      |   |   |   ENDIF
      |   |   ENDIF
      |   |   IF (t_cval_samps_PRI[di][nci] + samples_per_pulse[di] > 1025) &&\
      |   |   (PULSE_TYPE != c_spd_unmodulated) THEN
      |   |   |   Use f_cval_log_msg with index to message indicating\
      |   |   |   "ERROR PRF too low (di,nci), too many samples for pulse compression"
      |   |   |   IF there was an error writing the message THEN
      |   |   |   |   set the return_code to c_spd_failure
      |   |   |   ENDIF
      |   |   ENDIF
      |   |   IF (t_cval_samps_PRI[di][nci] > 1024) THEN
      |   |   |   Use f_cval_log_msg with index to message indicating\
      |   |   |   "ERROR PRF too low (di,nci), too many samples"
      |   |   |   IF there was an error writing the message THEN
      |   |   |   |   set the return_code to c_spd_failure
      |   |   |   ENDIF
      |   |   ENDIF
      |   ENDLOOP
      ENDLOOP
<---return the return_code
*/
}

/* end of cval_sat_pulse.c */
```

D-577

```
/*%%==========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence, 1992

File Name:      cval_sat_rra_param.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---  -----------  ------  ------------------------------------------------
n/a  07-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_rra_param

PURPOSE:        This function checks the RRA parameters of the Satellite and Radar data.


INPUT                                           SOURCE
-----                                           ------
sat_def                                         calling routine
num_msg                                         calling routine
cur_log_entry                                   calling routine

OUTPUT                                          DESTINATION
------                                          -----------
return_code                                     calling routine
num_msg                                         calling routine
cur_log_entry                                   calling routine

CREATION DATE:  June 04 1993

==========================================================================*/
#include "cval.h"

#define MAIN_DIAMETER     sat_def->antenna_model.main_antenna.RRA.main_diameter
#define MAIN_FOCAL_LEN    sat_def->antenna_model.main_antenna.RRA.main_focal_len
#define SUB_FOCAL_LEN     sat_def->antenna_model.main_antenna.RRA.sub_focal_len
#define SUB_DIAMETER      sat_def->antenna_model.main_antenna.RRA.sub_diameter
#define FEED_DIAMETER     sat_def->antenna_model.main_antenna.RRA.sub_diameter
#define ELEVATION         sat_def->antenna_model.main_antenna.RRA.attitude.elevation
#define ALTITUDE          sat_def->orbital_pos.altitude
```

```
f_cval_sat_rra_param(
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
    set the return code to t_spd_success
    IF sat_def->concept.concept == c_spd_RRA THEN
    |   magnify = MAIN_FOCAL_LEN / SUB_FOCAL_LEN
    |   spot_diam = MAIN_DIAMETER * SUB_FOCAL_LEN / MAIN_FOCAL_LEN
    |   IF sat_def->antenna_model.main_antenna.RRA.pattern_RSS_gen == c_spd_true THEN
    |   |   IF MAIN_DIAMETER < 10 * t_cval_lambda_min THEN
    |   |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   |   "WARNING RRA main beam very wide"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF MAIN_FOCAL_LEN > MAIN_DIAMETER THEN
    |   |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   |   "WARNING high RRA f/D ratio"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF MAIN_FOCAL_LEN < 0.25 * MAIN_DIAMETER THEN
    |   |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   |   "WARNING low RRA f/D ratio"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF (SUB_DIAMETER < t_cval_lambda_min) && (SUB_DIAMETER < spot_diam)) THEN
    |   |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   |   "WARNING RRA subreflector too small"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF SUB_DIAMETER > 0.5 * MAIN_DIAMETER THEN
    |   |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   |   "WARNING RRA subreflector too large"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF SUB_FOCAL_LEN > SUB_DIAMETER THEN
    |   |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   |   "WARNING high RRA subreflector f/D ratio"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF SUB_FOCAL_LEN < 0.25 * SUB_DIAMETER THEN
    |   |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   |   "WARNING low RRA subreflector f/D ratio"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF magnify < 2.0 THEN
    |   |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   |   "WARNING low RRA magnification"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF magnify > 50.0 THEN
    |   |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   |   "WARNING high RRA magnification"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   ENDIF
    |   IF FEED_DIAMETER < spot_diam THEN
    |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   "WARNING RRA feed too small, scanning is limited, blockage not correctly modelled"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    |   x = FEED_DIAMETER - spot_diam
    |   z = MAIN_FOCAL_LEN - SUB_FOCAL_LEN
    |   set the t_cval_RRA_scan_limit to atan(x/(z * magnify))
    |
    |   set Re to 6378
    |   set cosgraz = sin(ELEVATION) * (Re + ALTITUDE)/ Re
    |   IF cosgraz > 1 THEN
    |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   "WARNING main beam axis not on earth"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ELSE
    |   |   graz = (180/c_cval_pi) * acos(cosgraz)
    |   |   Use f_cval_log_msg with msg index to message indicating\
    |   |   "INFO main beam axis at graz 'graz' degrees grazing angle"
    |   |   IF there was an error writing the message THEN
    |   |   |   set the return_code to c_spd_failure
    |   |   ENDIF
    |   ENDIF
    ENDIF
<---return the return_code
*/
}

/* end of cval_sat_rra_param.c */
```

```
/*%%=========================================================================
SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence, 1992

File Name:        cval_sat_ssp.c

Identification:   [@(#)]

SCCS Id:          %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:         Chris Candler

Change History:

SCR   Date         Author   Change made
---   -----------  ------   ------------------------------------------------
n/a   07-JUN-1993  fcc      Initial Release

CSU_NAME:         CVAL

FUNCTION:         f_cval_sat_ssp

PURPOSE:          This function checks the SSP location.


INPUT                                                          SOURCE
-----                                                          ------
sat_def                                                        calling routine
num_msg                                                        calling routine
cur_log_entry                                                  calling routine

OUTPUT                                                         DESTINATION
------                                                         -----------
return_code                                                    calling routine
num_msg                                                        calling routine
cur_log_entry                                                  calling routine

CREATION DATE:  June 07 1993

===========================================================================*/
#include "cval.h"

f_cval_sat_ssp(
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry
)
{
/*|
    initialize the return_code to c_spd_success
    IF sat_def_ptr->orbital_pos.loc_method == c_spd_given_beam_loc_method THEN // 'derived'
    |   set SSP_loc to f_srun_init_ssp() // compute initial SSP longitude and latitude
    |   Use f_cval_log_msg with index to message indicating\
    |   "INFO derived SSP location: 'SSP_loc.lat' latitude, 'SSP_loc.lon' longitude"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
    IF SSP_loc.lat > (c_cval_pi/180) * sat_def_ptr->orbital_pos.incl_angle THEN // radians
    |   Use f_cval_log_msg with index to message indicating\
    |   "ERROR SSP not possible,greater than inclination"
    |   IF there was an error writing the message THEN
    |   |   set the return_code to c_spd_failure
    |   ENDIF
    ENDIF
<---return the return_code
*/
}

/* end of cval_sat_ssp.c */
```

```
/*%%=========================================================================
SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_sat_tap_delays.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR   Date        Author   Change made
---  -----------  ------   ------------------------------------------------
n/a  10-JUN-1993  fcc      Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_sat_tap_delays

PURPOSE:        This function validates the tap delays of the Satellite and Radar data.


INPUT                                                   SOURCE
-----                                                   ------
sat_def                                                 calling routine
num_msg                                                 calling routine
cur_log_entry                                           calling routine

OUTPUT                                                  DESTINATION
------                                                  -----------
return_code                                             calling routine
num_msg                                                 calling routine
cur_log_entry                                           calling routine
aperture_time                                           calling routine

CREATION DATE:  June 10 1993

==========================================================================*/
#include "cval.h"

#define CFA_APERT_X     sat_def->antenna_model.main_antenna.CFA.aperture.x
#define CFA_APERT_Y     sat_def->antenna_model.main_antenna.CFA.aperture.y
#define TAP_DELAY       sat_def->antenna_model.aux_antenna[aux].delay_taps[aux][tap]
#define NEXT_TAP_DELAY  sat_def->antenna_model.aux_antenna[aux].delay_taps[aux][s]

f_cval_sat_tap_delays(
```

D. [illegible]

```
    t_crad_sat_def *sat_def,
    t_spd_int32    *num_msg,
    char           **cur_log_entry,
    t_spd_real     *aperture_time
)
{
/*|
    initialize the return_code to c_spd_success
    IF sat_def->concept.concept == c_spd_RRA THEN
    |   aperture_size = sat_def->antenna_model.main_antenna.RRA.main_diameter
    ELSE
    |   aperture_size = sqrt((CFA_APERT_X ^ 2) + (CFA_APERT_Y ^ 2))
    ENDIF
    apert_time = aperture_size / 0.3          // nano seconds
    FOR aux = 0 to sat_def->antenna_model.aux_count - 1
    LOOP
    |   FOR tap = 0 to sat_def->antenna_model.aux_antenna[aux].tap_count - 1
    |   LOOP
    |   |   IF (abs(TAP_DELAY) < .05 * aperture_time) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "WARNING tap delay smaller than ideal (aux,tap)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   IF (abs(TAP_DELAY) > 5 * aperture_time) THEN
    |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   "WARNING tap delay larger than ideal (aux,tap)"
    |   |   |   IF there was an error writing the message THEN
    |   |   |   |   set the return_code to c_spd_failure
    |   |   |   ENDIF
    |   |   ENDIF
    |   |   FOR s = tap + 1 TO sat_def->antenna_model.aux_antenna[aux].tap_count - 1
    |   |   LOOP
    |   |   |   IF (abs(TAP_DELAY - NEXT_TAP_DELAY) < 0.05 * aperture_time) THEN
    |   |   |   |   Use f_cval_log_msg with index to message indicating\
    |   |   |   |   "WARNING tap delay difference smaller than ideal (aux,tap,s)"
    |   |   |   |   IF there was an error writing the message THEN
    |   |   |   |   |   set the return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   |   IF (abs(TA      Y - NEXT_TAP_DELAY) > 5 * aperture_time) THEN
    |   |   |   |   Use f_      og_msg with index to message indicating\
    |   |   |   |   "WAR         delay difference larger than ideal (aux,tap,s)"
    |   |   |   |   IF ther      an error writing the message THEN
    |   |   |   |   |   s        return_code to c_spd_failure
    |   |   |   |   ENDIF
    |   |   |   ENDIF
    |   |   ENDLOOP
    |   ENDLOOP
    ENDLOOP
<---return the return_coc
*/
}

/* end of cval_sat_tap_c          */
```

```
/*%%============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defence, 1992

File Name:      cval_log_msg.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Chris Candler

Change History:

SCR    Date       Author     Change made
---  -----------  ------  --------------------------------------------------
n/a  07-JUN-1993  fcc     Initial Release

CSU_NAME:       CVAL

FUNCTION:       f_cval_log_msg

PURPOSE:        This function logs the CVAL message to the val_log.


INPUT                                                   SOURCE
-----                                                   ------
num_msg                                                 calling routine
cur_log_entry                                           calling routine
msg_number                                              calling routine
msg_param0                                              calling routine
msg_param2                                              calling routine
msg_param3                                              calling routine
msg_param4                                              calling routine
msg_param5                                              calling routine
msg_param6                                              calling routine
msg_param7                                              calling routine

OUTPUT                                                  DESTINATION
------                                                  -----------
return_code                                             calling routine
num_msg                                                 calling routine
cur_log_entry                                           calling routine

CREATION DATE:  June 07 1993

==============================================================================*/
#include "cval.h"

```

```
f_cval_log_msg()
    /* Check the SSP location of the Satellite and Radar Data */
    /* (t_spd_int32 *)    num_msg */
    /* (char **)          cur_log_entry */
    /* t_spd_int16        msg_number    */
    /* t_spd_int16        msg_param0    */
    /* t_spd_int16        msg_param1    */
    /* t_spd_int16        msg_param2    */
    /* t_spd_int16        msg_param3    */
    /* t_spd_int16        msg_param4    */
    /* t_spd_int16        msg_param5    */
    /* t_spd_int16        msg_param6    */
    /* t_spd_int16        msg_param7    */
{
/*|
    initialize the return_code to c_spd_success
    allocate memory for the size of the message corresponding to the msg_number
    IF there was an error returned from the memory allocation THEN
    |   change the return_code to c_spd_failure
    ELSE
    |   point the cur_log_entry pointer to the new memory which was just allocated
    |   IF the message has parameters THEN
    |   |   construct the message with the message parameters
    |   |   Use strcpy to copy the message to the allocated memory
    |   ELSE
    |   |   Use strcpy to copy the message from the look-up table to the allocated memory
    |   ENDIF
    |   *num_msg++
    |   increment the cur_log_entry pointer
    ENDIF
<---return the return_code
*/
}

/* end of cval_log_msg.c */
```

```
                        TBD CROSS REFERENCE REPORT
=================================================================================================
PAGE NO.   LINE NO.         ENCLOSING FUNCTION


  NO TBDs TO REFERENCE
```

```
==========================================================================================================
 O.   LINE NO.


  1        48        f_cval_run_def
 89      4016        |    f_cval_log_msg
  5       215        |    f_cval_run_local
  9       374        |    f_cval_run_config
 89      4016        |    |    f_cval_log_msg
 12       497        |    f_cval_run_beam
 89      4016        |    |    f_cval_log_msg
 15       643        |    f_cval_run_blanker
 89      4016        |    |    f_cval_log_msg
 19       844        |    f_cval_run_norm_equ
 89      4016        |    |    f_cval_log_msg
 22      1033        |    f_cval_run_pulse_compr
 89      4016        |    |    f_cval_log_msg
 24      1129        |    f_cval_run_unambig_prf
 89      4016        |    |    f_cval_log_msg
 27      1226        |    f_cval_run_dpca
 89      4016        |    |    f_cval_log_msg
 30      1355        |    f_cval_run_non_coh
 89      4016        |    |    f_cval_log_msg
 32      1464        |    f_cval_run_cfar
 89      4016        |    |    f_cval_log_msg
 35      1632        |    f_cval_run_data_space
         4016        |    |    f_cval_log_msg

 40      1901        f_cval_sat_def
 83      3816        |    f_cval_sat_ssp
 89      4016        |    |    f_cval_log_msg
 64      2952        |    f_cval_sat_freq_range
 89      4016        |    |    f_cval_log_msg
 51      2359        |    f_cval_sat_cfa_grid
 89      4016        |    |    f_cval_log_msg
 81      3656        |    f_cval_sat_rra_param
 89      4016        |    |    f_cval_log_msg
 78      3492        |    f_cval_sat_pulse
 89      4016        |    |    f_cval_log_msg
 45      2126        |    f_cval_sat_beam
 89      4016        |    |    f_cval_log_msg
 48      2265        |    |    f_cval_sat_beam_point_val_tst
 62      2869        |    f_cval_sat_decimation
 89      4016        |    |    f_cval_log_msg
 57      2603        |    f_cval_sat_data_space
 54      2479        |    |    f_cval_sat_data_capt
 89      4016        |    |    f_cval_log_msg
 85      3901        |    f_cval_sat_tap_delays
 89      4016        |    |    f_cval_log_msg
 43      2003        |    f_cval_sat_aux_pos
 89      4016        |    |    f_cval_log_msg
 75      3400        |    f_cval_sat_pri
         4016        |    |    f_cval_log_msg
         3274        |    f_cval_sat_pc_pos
 89      4016        |    |    f_cval_log_msg
 69      3148        |    f_cval_sat_overlap
```

D-55

```
                                   INVOCATION TREE
==============================================================================================================
PAGE NO.   LINE NO.

   89        4016          |   |   f_cval_log_msg
   67        3070          |   f_cval_sat_gen_opt
   89        4016          |   |   f_cval_log_msg
```

```
                                   ERROR SUMMARY
===========================================================================================================
E NO.   LINE NO.    ERROR TEXT


NO ERRORS TO REPORT
```

D-E61

THIS PAGE INTENTIONALLY LEFT BLANK

D1.18 Error Logging (CERR)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      cerr_intf.h - Controller Utility External Include file

                Package CERR supports the error logging utility functions
                within the Controller functional area.

Identification: [@(#)]

SCCS Id:        @(#)cerr_intf.h 1.2 93/05/10

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file declares funtion description for the error logging
                service routines in the package CERR.

Designer:       Stephen Tran

Change History:

    SCR   Date         Author   Change made
    ---   -----------  ------  ---------------------------------------

    n/a  16-Feb-1993    cst    Initial Release

===========================================================================*/

#ifndef cerr_intf_h
#define cerr_intf_h

/*-------------------------------------------------------------------------- *
 *                                                                           *
 *    DEFINE CONSTANTS                                                       *
 *                                                                           *
 *--------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------- *
 *                                                                           *
 *    DEFINE ENUMERATED DATA TYPES                                           *
 *                                                                           *
 *--------------------------------------------------------------------------*/
/* the severity of the error message */

typedef enum {
  c_cerr_fatal,                /* fatal error */
  c_cerr_warning,              /* warning error */
  c_cerr_info                  /* information */
} t_cerr_severity_code;

/*-------------------------------------------------------------------------- *
 *                                                                           *
 *    DEFINE FUNCTION SPECIFICATION                                          *
 *                                                                           *
 *--------------------------------------------------------------------------*/

/*=========================================================================

   Function:    f_cerr_initialize

   Description:
      This function initializes the error log file named by pathname and
      return a stream with it.  pathname points to a character string
      that contains the directory and the name of file to be opened.
      If pathname is a NULL pointer, then it will assign the stream
      to stderr (standard error output). The stream will be used in
      subsequent operations.

   Arguments:
      pathname                  - in    - directory and file name of
                                          the error log.
                                          if NULL pointer is passed, then
                                          stream will be assigned to
                                          stderr.
      stream                    -   out - error log stream

   Exceptions:
      s_spd_failure is returned, if it failed to create error log file.

   Note:
      If the file named by pathname exists; then it will be deleted and
      re-created.

=============================================================================*/
extern t_spd_status f_cerr_initialize (
                char    *pathname,
                FILE    *stream);

/*=============================================================================

   Function:    f_cerr_log

   Description:
      This function writes error message into the named stream which
      was opened by f_cerr_initialize. The stream might be pointing
      to a file or stderr.

   Arguments:
      stream                    - in    - stream opened by f_cerr_initialize
      module_name               - in    - module name at which the error occurs
      severity_code             - in    - severity of error
      message                   - in    - error message

   Exceptions:
      s_spd_failure is returned, if it failed to write message to file or stderr

=============================================================================*/
extern t_spd_status f_cerr_log (
                FILE    *stream,
                char    *module_name,
                t_cerr_severity_code severity_code,
                char    *message);

/*=============================================================================

   Function:    f_cerr_close

   Description:
      This function closes the named stream which was opened by
      f_cerr_intialize.

   Arguments:
      stream                    - in    - stream opened by f_cerr_initialize

   Exceptions:
      s_spd_failure is returned, if the named stream can NOT be closed.

=============================================================================*/
extern t_spd_status f_cerr_close (
                FILE    *stream);

/*=============================================================================

   Function:    f_cerr_merge_files

```

```
    Description:
       This function merges more than one error log files into one output file.
       It merges files starting from input_pathname1 to input_pathname3 and
       it terminates when a NULL pointer is met. Therefore, it can merge
       either two files or three files into one output file.

    Arguments:
       output_pathname          - in    - pathname (directory and file name)
                                          of output file
       input_pathname1          - in    - pathname of input file #1
       input_pathname2          - in    - pathname of input file #2
       input_pathname3          - in    - pathname of input file #3

    Exceptions:
       s_spd_failure is returned, if either of input/output files
       can not be opened or it failed in reading error log file or
       writing output file.

=============================================================================*/
extern t_spd_status f_cerr_merge_files (
                char *output_pathanme,
                char *input_pathname1,
                char *input_pathname2,
                char *input_pathname3);

#endif  /* end of cerr_intf_h */
```

D-[illegible]

```
/*===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      cerr_func.c - Error logging utility functions

Identification: [@(#)]

SCCS Id:        @(#)cerr_func.c 1.2 93/05/10

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This file implements the error logging utility in the
                Controller functional area.

Designer:       Stephen Tran

Change History:

   SCR   Date      Author    Change made
   --- ----------- ------  -----------------------------------------
   n/a 29-Jan-1993  cst    Initial Release

===========================================================================*/

/* Included external declarations */
#include <sys/time.h>
#include <stdio.h>
#include "spd_intf.h"
#include "cerr_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* Implementation */
/*===========================================================================

    Function:    f_cerr_initialize

    Description:
       This function initializes the error log file named by pathname and
       return a stream with it.  pathname points to a character string
       that contains the directory and the name of file to be opened.
       If pathname is a NULL pointer, then it will assign the stream
       to stderr (standard error output). The stream will be used in
       subsquent read/write operations.  If the error file exists, then
       it will be re-opened as an empty file.

    Arguments:
       pathname                  - in    - directory and file name of
                                           the error log.
                                           if NULL pointer is passed, then
                                           stream will be assigned to
                                           stderr.
       stream                    -   out - error log stream

    External variables:
       none

    Exceptions:
       s_spd_failure returned if it can NOT open file or it can NOT delete
       the existing file.

   Notes:
      none

   High Level PDL
      BEGIN  f_cerr_initialize
        IF pathname == NULL THEN
           stream = stderr
        ELSE
           USE stat TO check to see whether the file exists or not
           IF file exists THEN
              IF write permission is on THEN
                 USE unlink TO remove the existing error file
              ELSE
                 return s_spd_failure
              ENDIF
           ENDIF

           USE fopen TO open a new error file
           IF error THEN return s_spd_failure
        ENDIF
      END f_cerr_initialize
==========================================================================*/
t_spd_status f_cerr_initialize (
                char    *pathname,
                FILE    *stream)
{
}

/*==========================================================================

   Function:    f_cerr_log

   Description:
      This function writes error message into the named stream which
      was opened by f_cerr_initialize. This file stream might be pointing
      to a file or stderr.

   Arguments:
      stream                      - in    - stream opened by f_cerr_initialize
      module_name                 - in    - module name at which the error occurs
      severity_code               - in    - severity of error
      message                     - in    - error message

   External variables:
      none

   Exceptions:
      s_spd_failure returned, if it can NOT write the message to file or stderr

   Notes:
      none

   High Level PDL
      BEGIN f_cerr_log
        USE time TO get the current system time
        USE asctime TO get the time string
        format error message using time, severity, message
        USE fwrite TO the error message to file or stderr
        USE fflush TO flush the named stream
        IF error THEN return s_spd_failure
      END f_cerr_log
==========================================================================*/
t_spd_status f_cerr_log (
                FILE    *stream,
                char    *module_name,
                t_cerr_severity_code severity_code,
                char    *message)
{
}
/*==========================================================================

    Function:    f_cerr_close

    Description:
       This function closes the named stream which was opened by
       f_cerr_intialize.

    Arguments:
       stream                        - in    - stream opened by f_cerr_initialize

    External variables:
       none

    Exceptions:
       s_spd_failure returned, if the named stream can NOT be closed.

    Notes:
       None.

    High Level PDL
       BEGIN f_cerr_close
         USE fclose to close or flush the named stream
         IF error THEN return s_spd_failure
       END f_cerr_close
=============================================================================*/
t_spd_status f_cerr_close (
                FILE    *stream)
{
}

/*=============================================================================

    Function:    f_cerr_merge_files

    Description:
       This function merges more than one error log files into one output file.
       It merges files starting from input_pathname1 to input_pathname3 and
       it terminates when a NULL pointer is met. Therefore, it can merge
       either two files or three files into one output file.

    Arguments:
       output_pathname            - in    - pathname (directory and file name)
                                            of output file
       input_pathname1            - in    - pathname of input file #1
       input_pathname2            - in    - pathname of input file #2
       input_pathname3            - in    - pathname of input file #3

    Exceptions:
       s_spd_failure is returned, if either of input/output files
       can not be opened or it failed in reading error log file or
       writing output file.

    High Level PDL
       BEGIN f_cerr_merge_files
         USE creat TO open output_pathname with permission 0644
         FOR all input files passed by the argument list
            USE open TO open input file
            DO UNTIL EOF
              USE read TO read input file
              USE write TO write to output file
            ENDDO
            USE close TO close input file
         ENDFOR
         USE close to close the output file
         IF error THEN return s_spd_failure
       END f_cerr_merge_files
=============================================================================*/
extern t_spd_status f_cerr_merge_files (
                char *merge_pathanme,
                char *input_pathname1,
                char *input_pathname2,
                char *input_pathname3);
```

THIS PAGE INTENTIONALLY LEFT BLANK

D1.19 System Services (CSYS)

THIS PAGE INTENTIONALLY LEFT BLANK

```

/*===========================================================================

SPD - Signal Processing Demonstrator Computer SPD System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csys_genshdwn.c - Main program of generation shutdown

Identification: [@(#)]

SCCS Id:        @(#)csys_genshdwn.c     1.1 93/04/22

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package implements the main program to shutdown
                the Controller Generation Server and RSS Generation
                Manager.


Designer:       Stephen Tran

Change History:

   SCR  Date        Author   Change made
   ---  ----------  ------  ---------------------------------------
   ??? 10-Mar-1993  cst     Initial Release

===========================================================================*/

/* Included external declarations */
#include <rpc/rpc.h>
#include "spd_intf.h"
#include "cget_intf.h"
#include "crgi_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* Implementation */
/*===========================================================================

   program:     csys_genshdwn

   Description:
      Main program to shutdown generation servers: Controller Generation
      Server and RSS Generation Manager.

      This program checks to see whether both Generation Server and
      Generation Manager is running or not. This program does nothing
      if both servers are down; otherwise, this program will shutdown
      the one alive.

   Arguments:
      none

   External variables:
      none

   Exceptions:
      none

   Notes:
      Assumption is made that the SPD System Manager spawns both Controller
      Generation Server and RSS Generation Manager; therefore, only
```

D-375

```
     the SPD System Manager can shutdown these processes.
     As a rule, we should shutdown the Controller Generation Server,
     then RSS Gneration Manager

  High Level PDL
     BEGIN csys_genshdwn
        get node names of Controller and RSS from environment variables:
            CNTL_NODE and RSS_NODE
        gensvr_died = FALSE
        genmgr_died = FALSE
        USE clnt_create TO create client connection to Controller Generation
            Server
        IF error THEN
           gensvr_died = TRUE
           USE clnt_call TO send a ping message to Controller Generation Server
           IF no answer after timeout THEN
              gensvr_died = TRUE
           ENDIF
        ENDIF

        USE clnt_create TO create client connection to RSS Generation Manager
        IF error THEN
           genmgr_died = TRUE
           USE clnt_call TO send a ping message to RSS Generation Manager
           IF no answer after timeout THEN
              genmgr_died = TRUE
           ENDIF
        ENDIF

        IF (gensvr_died == FALSE)
           USE clnt_call TO send a status request message to
             Controller Generation Server
           IF generation activity is currently running THEN
              prompt user for confirmation whether to kill the current run
           ENDIF
           IF (generation activity is currently idle) or
              (the user answers to kill the current run) THEN
              USE clnt_call TO send abort generation message to
                Controller Generation Server
           ENDIF
           USE clnt_call TO send shutdown message to
             Controller Generation Manager
           gensvr_died = TRUE
        ELSE IF (genmgr_died == FALSE) and (gensvr_died == TRUE) THEN
           USE clnt_call TO send a status request message to
             RSS Generation Manager
           IF generation activity is currently running in RSS THEN
              prompt user for confirmation whether to kill the current run
           ENDIF
           IF  (the user answers to kill the current run) THEN
              USE clnt_call TO send abort generation message to
                RSS Generation Manager
           ENDIF
           USE clnt_call TO send shutdown message to
             RSS Generation Manager
           genmgr_died = TRUE
        ENDIF

        IF (genmgr_died == FALSE) THEN
           USE clnt_call TO send shutdown message to
             RSS Generation Manager
        ENDIF

     END csys_genshdwn
  End of PDL
==========================================================================*/

/* Routine Declaration */
t_spd_status main (
      t_spd_int32       argc,
        char                 **argv,
        char                 **envp)
{   /* Begin of main */
}   /* End of main */
```

```

/*===========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csys_genstart.c - Main program of generation startup

Identification: [@(#)]

SCCS Id:        @(#)csys_genstart.c     1.1 93/04/22

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package implements the main program to startup
                the Controller Generation Server and RSS Generation
                Manager.


Designer:       Stephen Tran

Change History:

   SCR  Date        Author   Change made
   ---  ----------- ------   ---------------------------------------

   ??? 10-Mar-1993  cst      Initial Release

===========================================================================*/

/* Included external declarations */
#include <rpc/rpc.h>
#include "spd_intf.h"
#include "cget_intf.h"
#include "crgi_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* Implementation */
/*===========================================================================

   program:     csys_genstart

   Description:
      Main program to startup generation servers: Controller Generation
      Server and RSS Generation Manager.

      This program checks to see whether both Generation Server and
      Generation Manager is running or not. This program does nothing
      if both servers are up.  If either of them was not running, this
      program will shutdown the one alive to avoid the case they are
      not in sync.

      This program will then launch both Generation Server and RSS Generation
      Manager via use of "rsh" using the username of the System Manager.
      We assume that the user has remote log-in permission as the System
      Manager, i.e. the user name should be found in the System Manager's
      .rhosts file.

   Arguments:
      none

   External variables:
      none
```

```

Exceptions:
   none

Notes:
   As a rule, we should startup RSS Gneration Manager, then the Controller
   Generation Server. However, we should perform shutdown procedure
   in the reverse order.

High Level PDL
   BEGIN csys_genstart
      get node names of Controller and RSS from environment variables:
          CNTL_NODE and RSS_NODE
      gensvr_died = FALSE
      genmgr_died = FALSE
      USE clnt_create TO create client connection to Controller Generation
          Server
      IF error THEN
         gensvr_died = TRUE
         USE clnt_call TO send a ping message to Generation Server
         IF no answer after timeout THEN
            gensvr_died = TRUE
         ENDIF
      ENDIF

      USE clnt_create TO create client connection to RSS Generation Manager
      IF error THEN
         genmgr_died = TRUE
         USE clnt_call TO send a ping message to Generation Manager
         IF no answer after timeout THEN
            genmgr_died = TRUE
         ENDIF
      ENDIF

      IF (gensvr_died == FALSE) and (genmgr_died == FALSE) THEN
         prompt user that both Controller Generation Server and
         RSS Generation Manager are up already
         exit
      ENDIF

      IF (gensvr_died == TRUE) and (genmgr_died == FALSE) THEN
         USE clnt_call TO send a status request message to RSS Generation
           Manager
         IF generation activity is currently running THEN
            prompt user for confirmation to kill the current run
         ENDIF
         IF (generation activity is currently idle) or
            (the user answers to kill the current run) THEN
            USE clnt_call TO send shutdown message to RSS Generation Manager
            genmgr_died = TRUE
         ENDIF
      ENDIF
      IF (gensvr_died == FALSE) and (genmgr_died == TRUE) THEN
         USE clnt_call TO send shutdown message to Controller
           Generation Server
         gensvr_died = TRUE
      ENDIF

      IF (gensvr_died = TRUE) and (genmgr_died = TRUE) THEN
         USE "rsh -l spdmgr -n $CNTL_NODE" TO spawn Controller
           Generation Server
         USE "rsh -l spdmgr -n $RSS_NODE"  TO spawn RSS Generation Manager

         USE clnt_create TO create client connection to both servers
         USE clnt_call TO send ping message to both servers
         IF no response THEN
            log communucation error
         ENDIF
      ENDIF

```

```
        END csys_genstart
     End of PDL
==========================================================================*/

/* Routine Declaration */
t_spd_status main (
        t_spd_int32       argc,
        char              **argv,
        char              **envp)
{   /* Begin of main */
}   /* End of main */
```

```

/*=========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csys_simshdwn.c - Main program of simulation shutdown

Identification: [@(#)]

SCCS Id:        @(#)csys_simshdwn.c     1.1 93/04/22

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package implements the main program to shutdown
                the Controller Simulation Server and RSS Simulation
                Manager.


Designer:       Stephen Tran

Change History:

   SCR  Date         Author   Change made
   ---  -----------  ------   ----------------------------------------

   ??? 10-Mar-1993  cst      Initial Release

=========================================================================*/

/* Included external declarations */
#include <rpc/rpc.h>
#include "spd_intf.h"
#include "csit_intf.h"
#include "crpi_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* Implementation */
/*=========================================================================

   program:     csys_simstart

   Description:
      Main program to shutdown simulation servers: Controller Simulation
      Server and RSS Playback Manager.

      This program checks to see whether both Controller Simulation Server
      and RSS Playback Manager is running or not. This program does nothing
      if both servers are down; otherwise, this program will shutdown
      the one alive.

   Arguments:
      none

   External variables:
      none

   Exceptions:
      none

   Notes:
      none

```

D-[illegible]

```
   High Level PDL
      BEGIN csys_simstart
         get node names of Controller and RSS from environment variables:
             CNTL_NODE and RSS_NODE
         simsvr_died = FALSE
         simmgr_died = FALSE
         USE clnt_create TO create client connection to Controller Simulation
             Server
         IF error THEN
            simsvr_died = TRUE
            USE clnt_call TO send a ping message to Controller Simulation Server
            IF no answer after timeout THEN
               simsvr_died = TRUE
            ENDIF
         ENDIF

         USE clnt_create TO create client connection to RSS Playback Manager
         IF error THEN
            simmgr_died = TRUE
            USE clnt_call TO send a ping message to RSS Playback Manager
            IF no answer after timeout THEN
               simmgr_died = TRUE
            ENDIF
         ENDIF

         IF (simsvr_died == FALSE)
            USE clnt_call TO send a status request message to
              Controller Simulation Server
            IF simulation activity is currently running THEN
               prompt user for confirmation whether to kill the current run
            ENDIF
            IF (simulation activity is currently idle) or
               (the user answers to kill the current run) THEN
               USE clnt_call TO send abort simulation message to
                 Controller Simulation Server
            ENDIF
            USE clnt_call TO send shutdown message to
              Controller Simulation Server
            simsvr_died = TRUE
         ELSE IF (simmgr_died == FALSE) and (simsvr_died == TRUE) THEN
            USE clnt_call TO send a status request message to
              RSS Playback Manager
            IF simulation activity is currently running in RSS THEN
               prompt user for confirmation whether to kill the current run
            ENDIF
            IF  (the user answers to kill the current run) THEN
               USE clnt_call TO send abort simulation message to
                 RSS Playback Manager
            ENDIF
            USE clnt_call TO send shutdown message to
              RSS Playback Manager
            simmgr_died = TRUE
         ENDIF
         IF (simmgr_died == FALSE) THEN
            USE clnt_call TO send shutdown message to
              RSS Playback Manager
         ENDIF

      END csys_simstart
   End of PDL
===========================================================================*/

/* Routine Declaration */
t_spd_status main (
      t_spd_int32        argc,
      char               **argv,
      char               **envp)
{   /* Begin of main */
}   /* End of main */
```

```

/*============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      csys_simstart.c - Main program of simulation startup

Identification: [@(#)]

SCCS Id:        @(#)csys_simstart.c     1.1 93/04/22

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package implements the main program to startup
                the Controller Simulation Server and RSS Simulation
                Manager.


Designer:       Stephen Tran

Change History:

   SCR  Date        Author   Change made
   ---  ----------- ------  ----------------------------------------
   ??? 10-Mar-1993  cst     Initial Release

===========================================================================*/

/* Included external declarations */
#include <rpc/rpc.h>
#include "spd_intf.h"
#include "csit_intf.h"
#include "crpi_intf.h"
/* End of Included external declarations */

/* Included internal declarations */
/* none */
/* End of Included internal declarations */

/* Implementation */
/*============================================================================

   program:     csys_simstart

   Description:
      Main program to startup simulation servers: Controller Simulation
      Server and RSS Playback Manager.

      This program checks to see whether both Simulation Server and
      Playback Manager is running or not. This program does nothing
      if both servers are up.  If either of them was not running, this
      program will shutdown the one alive to avoid the case they are
      not in sync.

      This program will then launch both Simulation Server and RSS Simulation
      Manager via use of "rsh" using the username of the System Manager.
      We assume that the user has remote log-in permission as the System
      Manager, i.e. the user name should be found in the System Manager's
      .rhosts file.

   Arguments:
      none

   External variables:
      none
```

```

  Exceptions:
     none

  Notes:
     none

  High Level PDL
     BEGIN csys_simstart
        get node names of Controller and RSS from environment variables:
            CNTL_NODE and RSS_NODE
        simsvr_died = FALSE
        simmgr_died = FALSE
        USE clnt_create TO create client connection to Controller Simulation
            Server
        IF error THEN
           simsvr_died = TRUE
           USE clnt_call TO send a ping message to Controller Simulation Server
           IF no answer after timeout THEN
              simsvr_died = TRUE
           ENDIF
        ENDIF

        USE clnt_create TO create client connection to RSS Playback Manager
        IF error THEN
           simmgr_died = TRUE
           USE clnt_call TO send a ping message to RSS Playback Manager
           IF no answer after timeout THEN
              simmgr_died = TRUE
           ENDIF
        ENDIF

        IF (simsvr_died == TRUE) and (simmgr_died == FALSE) THEN
           USE clnt_call TO send a status request message to RSS Playback
             Manager
           IF playback  is currently running THEN
              prompt user for confirmation to kill the current run
           ENDIF
           IF (playback is currently idle) or
              (the user answers to kill the current run) THEN
              USE clnt_call TO send shutdown message to Playback Manager
              simmgr_died = TRUE
           ENDIF
        ENDIF

        IF (simsvr_died == FALSE) and (simmgr_died == TRUE) THEN
           USE clnt_call TO send shutdown message to Simulation Server
           simsvr_died = TRUE
        ENDIF

        IF (simsvr_died == TRUE) and (simmgr_died == TRUE) THEN
           USE "rsh -l spdmgr -n $RSS_NODE"  TO spawn RSS Playback Manager
           USE "rsh -l spdmgr -n $CNTL_NODE" TO spawn Controller
             Simulation Server

           USE clnt_create TO create client connection to both servers
           USE clnt_call TO send ping message to both servers
           IF no response THEN
              log communication error
           ENDIF
        ENDIF

      END csys_simstart
   End of PDL
=========================================================================*/

/* Routine Declaration */
t_spd_status main (
      t_spd_int32       argc,
      char              **argv,
        char                **envp)
{   /* Begin of main */
}   /* End of main */
```

THIS PAGE INTENTIONALLY LEFT BLANK

D1.20 User Interface Services (CUIS)

THIS PAGE INTENTIONALLY LEFT BLANK

```
/*==========================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      UxXt.h - Basic functions provided by UIM/X config release

                the H file (UxXt.h) in UIM/X config release was
                copied and repackaged as UxXt.h

Identification: [@(#)]

SCCS Id:        @(#)UxXt.h      1.2 93/05/10

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package was taken from the UIM/X config release,
                (/apps/pvi/waveuimx/bin.sun4/config/UxXt.h), and is
                the include file for corresponding package UxXt.c.

                Under SPD system, the C file (UxXt.c) was repackaged as
                cuis_uxxt.c; this is not possible for the .h file, however,
                since UxXt.h is auto matically #included in every C file
                generated through uxcgen -x.

                None of the functions defined in UxXt.c and declared in
                in UxXt.h comply to the SPD naming conventions.

   Exported functions declared:

                UxPopupInterface()
                UxPopdownInterface()
                UxDestroyInterface()
                UxPutContext()
                UxGetContext()
                UxFreeClientDataCB()
                UxLoadResources()

Designer:       Visual Edge Software Ltd.
                (repackaged by Stephen Tran)

Change History:

    SCR    Date        Author    Change made
    ---  -----------   ------  -------------------------------------
    n/a  10-Mar-1993    cst    Initial Release

==========================================================================*/
/****************************************************************************/
/***                                                                     ***/
/***             Copyright (c) 1991, Visual Edge Software Ltd.           ***/
/***                                                                     ***/
/***   ALL RIGHTS RESERVED.  This notice is  intended  as  a  precaution ***/
/***   against  inadvertent publication, and shall not be deemed to con- ***/
/***   stitute an acknowledgment that publication has  occurred  nor  to ***/
/***   imply  any  waiver  of confidentiality.  The year included in the ***/
/***   notice is the year of the creation of the work.                   ***/
/***                                                                     ***/
/****************************************************************************/
/*                              UxXt.h                                     */
/****************************************************************************/

#ifndef _UX_XT_H_
#define _UX_XT_H_

```

```
/* The following macros are used in converting string values to the form
   required by the widgets */

#define RES_CONVERT( res_name, res_value) \
        XtVaTypedArg, (res_name), XmRString, (res_value), strlen(res_value) + 1

#define UxPutStrRes( wgt, res_name, res_value ) \
        XtVaSetValues( wgt, RES_CONVERT( res_name, res_value ), NULL )


#ifndef UX_INTERPRETER  /* Omit this section when interpreting the code */

/* The following macros are supplied for compatibility with swidget code */
#define swidget                 Widget
#define UxWidgetToSwidget(w)    (w)
#define UxGetWidget(sw)         (sw)
#define UxIsValidSwidget(sw)    ((sw) != NULL)

#define no_grab                 XtGrabNone
#define nonexclusive_grab       XtGrabNonexclusive
#define exclusive_grab          XtGrabExclusive


/* The following global variables are defined in the main() function */
extern  XtAppContext    UxAppContext;
extern  Widget          UxTopLevel;
extern  Display         *UxDisplay;
extern  int             UxScreen;


/* The following are error codes returned by the functions in UxXt.c */
#define UX_ERROR            -1
#define UX_NO_ERROR         0


/* The following are declarations of the functions in UxXt.c */

#ifdef _NO_PROTO

extern  int     UxPopupInterface();
extern  int     UxPopdownInterface();
extern  int     UxDestroyInterface();
extern  int     UxPutContext();
extern  caddr_t UxGetContext();
extern  void    UxFreeClientDataCB();
extern  void    UxLoadResources();

#else

extern  int     UxPopupInterface( Widget wgt, XtGrabKind grab_flag );
extern  int     UxPopdownInterface( Widget wgt );
extern  int     UxDestroyInterface( Widget wgt);
extern  int     UxPutContext( Widget wgt, caddr_t context );
extern  caddr_t UxGetContext( Widget wgt );
extern  void    UxFreeClientDataCB( Widget wgt, XtPointer client_data,
                                    XtPointer call_data );
extern  void    UxLoadResources( char *fname );

#endif /* _NO_PROTO */

#endif /* ! UX_INTERPRETER */

#endif /* ! _UX_XT_H_ */
```

L 340

```
/*============================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada
as represented by the Minister of National Defence

File Name:      cuis_uxxt.c - Basic set of Ux functions

                the C file (UxXt.c) in UIM/X config release was
                repackaged as cuis_uxxt.c

Identification: [@(#)]

SCCS Id:        @(#)cuis_uxxt.c 1.2 93/05/10

Responsible Organization:

                MacDonald Dettwiler and Associates

Purpose:        This package was taken from the UIM/X config release,
                (/apps/pvi/waveuimx/bin.sun4/config/UxXt.c), and
                provides the basic set of routines needed to link
                the SPD HMI codes produced through UIM/X and uxcgen.

                The functions defined here are only X/Xt dependent,
                hence UxXt.c is the only file file needed to make
                the application truly UIM/X-independent.

                None of the functions defined in UxXt.c and declared in
                in UxXt.h comply to the SPD naming conventions.

   Exported functions declared:

                UxPopupInterface()
                UxPopdownInterface()
                UxDestroyInterface()
                UxPutContext()
                UxGetContext()
                UxFreeClientDataCB()
                UxLoadResources()

Designer:       Visual Edge Software Ltd.
                (repackaged by Stephen Tran)

Change History:

   SCR  Date          Author  Change made
   ---  -----------   ------  -------------------------------------------

   n/a  10-Mar-1993   cst     Initial Release

============================================================================*/
/*************************************************************************
 * $Date: 91/11/05 19:24:38 $
 * $Revision: 1.2.33.2 $
 *
 * Copyright (c) 1991, Visual Edge Software Ltd.
 * ---------------------------------------------
 * ALL RIGHTS RESERVED.  This notice is  intended  as  a  precaution
 * against  inadvertent publication,  and shall not be deemed to
 * constitute an acknowledgment that publication has occurred nor to
 * imply any waiver of confidentiality. The year included in the
 * notice is the year of the creation of the work.
 *------------------------------------------------------------------------
 *                              UxXt.c
 *-----------------------------------------------------------------------*/

#include <stdio.h>

#include <X11/Intrinsic.h>
```

```
#include <X11/Shell.h>
#include <X11/StringDefs.h>
#include <X11/Xutil.h>
#include <X11/X.h>
#include <X11/Xmd.h>
#include <X11/Xlib.h>
#include <Xm/Xm.h>
#include <Xm/DialogS.h>

#include "UxXt.h"

static  XContext        xcontext_id = NULL;

/******************************************************************************
NAME:           GetTrueToplevel( wgt )

INPUT:          Widget  wgt             - the top user-created widget

RETURN:         Widget                  - the shell widget

DESCRIPTION:    Returns the true toplevel of that interface.
                This is needed for implicit shells where the widget passed to
                one of the Ux functions is not always the shell.

CREATION:       Visual Edge Software            April 6 1991
-----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
static  Widget  GetTrueToplevel( wgt )
        Widget  wgt;
#else
static  Widget  GetTrueToplevel( Widget wgt )
#endif
{
        if (wgt && !XtIsShell(wgt))
        {
                wgt = XtParent(wgt);
        }
        return wgt;
}

/******************************************************************************
NAME:           handle_dialog_child( wgt, manage_func )

INPUT:          Widget  wgt                     - the dialogShellWidget
                void    (*manage_func)()        - either XtManageChild
                                                  or XtUnmanageChild

RETURN:         int                             - UX_NO_ERROR if successfull
                                                  UX_ERROR otherwise

DESCRIPTION:    Handles the popping up or popping down of dialog shells
                by managing or unmanaging their children.

CREATION:       Visual Edge Software            Sept 19/91
-----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
static  int     handle_dialog_child( wgt, manage_func )
        Widget  wgt;
        void    (*manage_func)();
#else
static  int     handle_dialog_child( Widget wgt, void (*manage_func)() )
#endif
{
        int     i, num_children;
        Widget  *children;

        XtVaGetValues( wgt,
                        XmNnumChildren, &num_children,
                        XmNchildren, &children,
                        NULL );

        /* We manage/unmanage the first rectObj child in the list.
         * Note that the check for rectObjClass is necessary since some
         * implementations of Motif add protocol children to the dialogShell.
         */

        for (i = 0; i < num_children; i++)
        {
                if ( XtIsSubclass( children[i], rectObjClass ) )
                {
                        (*manage_func)(children[i]);
                        return ( UX_NO_ERROR );
                }
        }

        return ( UX_ERROR );
}

/******************************************************************************
NAME:           popup_dialog( wgt, grab_flag )

INPUT:          Widget          wgt             - dialogShell to pop up
                XtGrabKind      grab_flag       - the grab flag

RETURN:         void

DESCRIPTION:    Pops up a dialogShell.

CREATION:       Visual Edge Software            Sept 19/91
-----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
static  void    popup_dialog( wgt, grab_flag )
        Widget          wgt;
        XtGrabKind      grab_flag;
#else
static  void    popup_dialog( Widget wgt, XtGrabKind grab_flag )
#endif
{
        if ( handle_dialog_child( wgt, XtManageChild ) == UX_ERROR )
                XtPopup( wgt, grab_flag );
}

/******************************************************************************
NAME:           UxPopupInterface( wgt, grab_flag )

INPUT:          Widget          wgt             - Widget to popup
                XtGrabKind      grab_flag       - grab flag

RETURN:         int                     UX_ERROR or UX_NO_ERROR

DESCRIPTION:    Popups up an interface. The widget should be a toplevel widget.
                Note that special handling is required for dialogShells since
                those are popped up by managing their children if they have
                some.
                The grab_flag could be any of:
                                no_grab (XtGrabNone)
                                nonexclusive_grab (XtGrabNonexclusive)
                                exclusive_grab (XtGrabExclusive)

CREATION:       Visual Edge Software            April 6 1991
-----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
int     UxPopupInterface( wgt, grab_flag )
        Widget          wgt;
        XtGrabKind      grab_flag;
#else
int     UxPopupInterface( Widget wgt, XtGrabKind grab_flag )
#endif
{
        if (!(wgt = GetTrueToplevel(wgt)))
```

```
                return ( UX_ERROR );

        if ( XtIsSubclass( wgt, xmDialogShellWidgetClass ) )
        {
                popup_dialog( wgt, grab_flag );
        }
        else
        {
                XtPopup( wgt, grab_flag );
        }

        return ( UX_NO_ERROR );
}

/******************************************************************************
NAME:           popdown_dialog( wgt )

INPUT:          Widget  wgt             - dialogShell to popdown

RETURN:         void

DESCRIPTION:    Pops down a dialogShell.

CREATION:       Visual Edge Software            Sept 19/91
-----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
static  void    popdown_dialog( wgt )
        Widget  wgt;
#else
static  void    popdown_dialog( Widget wgt )
#endif
{
        if ( handle_dialog_child( wgt, XtUnmanageChild ) == UX_ERROR )
                XtPopdown( wgt );
}

/******************************************************************************
NAME:           UxPopdownInterface( wgt )

INPUT:          Widget  wgt             - Widget to popdown

RETURN:         int                     UX_ERROR / UX_NO_ERROR

DESCRIPTION:    Pops down an interface. The widget should be a toplevel widget.
                Note that special handling is required for dialogShells since
                those are popped down by unmanaging their children if they have
                some.

CREATION:       Visual Edge Software            April 6 1991
-----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
int     UxPopdownInterface( wgt )
        Widget  wgt;
#else
int     UxPopdownInterface( Widget wgt )
#endif
{
        if (!(wgt = GetTrueToplevel(wgt)))
                return ( UX_ERROR );

        if ( XtIsSubclass( wgt, xmDialogShellWidgetClass ) )
        {
                popdown_dialog( wgt );
        }
        else
        {
                XtPopdown( wgt );
        }

        return ( UX_NO_ERROR );
```

```
}

/******************************************************************************
NAME:           UxDestroyInterface( wgt )

INPUT:          Widget  wgt             - Widget to destroy

RETURN:         int                     UX_ERROR / UX_NO_ERROR

DESCRIPTION:    Destroys an interface.  The widget should be a toplevel widget.

CREATION:       Visual Edge Software            April 6 1991
----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
int     UxDestroyInterface( wgt )
        Widget  wgt;
#else
int     UxDestroyInterface( Widget wgt )
#endif
{
        if (!(wgt = GetTrueToplevel(wgt)))
                return ( UX_ERROR );

        XtDestroyWidget (wgt);

        return ( UX_NO_ERROR );
}

/******************************************************************************
NAME:           DeleteContextCB( wgt, client_data, call_data )

INPUT:          Widget          wgt             - widget causing the callback
                XtPointer       client_data     - not used
                XtPointer       call_data       - not used

RETURN:         void

DESCRIPTION:    Deletes the X context entry.

EXT REFERENCES: UxTopLevel, xcontext_id

CREATION:       Visual Edge Software            April 6 1991
----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
static  void    DeleteContextCB( wgt, client_data, call_data )
        Widget          wgt;
        XtPointer       client_data, call_data;
#else
static  void    DeleteContextCB( Widget wgt, XtPointer client_data,
                                                XtPointer call_data )
#endif
{
        (void) XDeleteContext( XtDisplay( UxTopLevel ),
                               (Window) wgt,
                               xcontext_id );
}

/******************************************************************************
NAME:           UxPutContext( wgt, context )

INPUT:          Widget  wgt             - Widget
                caddr_t context         - context pointer

RETURN:         int                     UX_ERROR / UX_NO_ERROR

DESCRIPTION:    Uses the X Context manager to store the given context pointer
                in a memory location that is indexed by the given widget id.
                Also adds a destroyCallback to delete that context when the
                widget is destroyed.

```

```
EXT REFERENCES: UxTopLevel, xcontext_id
EXT EFFECTS:    xcontext_id

CREATION:       Visual Edge Software            April 6 1991
----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
int     UxPutContext( wgt, context )
        Widget  wgt;
        caddr_t context;
#else
int     UxPutContext( Widget wgt, caddr_t context )
#endif
{
        int             status;

        if ( xcontext_id == NULL )
                xcontext_id = XUniqueContext();

        if ( wgt == NULL )
                return ( UX_ERROR );

        status = XSaveContext( XtDisplay( UxTopLevel ),
                               (Window) wgt,
                               xcontext_id,
                               context );
        if ( status != 0 )
                return ( UX_ERROR );

        XtAddCallback (wgt, XmNdestroyCallback, DeleteContextCB, NULL);

        return ( UX_NO_ERROR );
}

/******************************************************************************
NAME:           UxGetContext( wgt )

INPUT:          Widget  wgt             - widget

RETURN:         caddr_t                 - the context pointer

DESCRIPTION:    Uses the X Context manager to find the context pointer
                stored in a memory location indexed by the given widget id.

EXT REFERENCES: UxTopLevel, xcontext_id

CREATION:       Visual Edge Software            April 6 1991
----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
caddr_t UxGetContext( wgt )
        Widget  wgt;
#else
caddr_t UxGetContext( Widget wgt )
#endif
{
        int     status;
        caddr_t context;

        if ( wgt == NULL )
                return ( (caddr_t) NULL );

        status = XFindContext( XtDisplay( UxTopLevel ),
                               (Window) wgt,
                               xcontext_id,
                               &context );

        if ( status != 0 )
                return  ( (caddr_t) NULL );

        return ( context );
}
```

```

/******************************************************************************
NAME:           UxFreeClientDataCB( wgt, client_data, call_data )

INPUT:          Widget          wgt             - widget
                XtPointer       client_data     - pointer to be freed
                XtPointer       call_data       - not used

RETURN:         void

DESCRIPTION:    This callback function simply frees the client data.

CREATION:       Visual Edge Software            April 6 1991
-----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
void    UxFreeClientDataCB( wgt, client_data, call_data )
        Widget          wgt;
        XtPointer       client_data, call_data;
#else
void    UxFreeClientDataCB( Widget wgt, XtPointer client_data,
                                XtPointer call_data )
#endif
{
        if (client_data != NULL)
                XtFree(client_data);
}

/******************************************************************************
NAME:           UxLoadResources( fname )

INPUT:          char    *fname          - name of resource file

RETURN:         void

DESCRIPTION:    This function is provided as a stub function to load the
                resource file that was generated by the C Code Generator
                into the current toolkit resource database.
                A call to this function is automatically generated with the
                appropriate resource filename when resources are set to Public.

CREATION:       Visual Edge Software            April 6 1991
-----------------------------------------------------------------------------*/
#ifdef _NO_PROTO
void    UxLoadResources( fname )
        char    *fname;
#else
void    UxLoadResources( char *fname )
#endif
{
}

/***END*OF*FILE***************************************************************/

```

APPENDIX E

RSS PDL

By: M. Baskin (RCL)

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

Page

APPENDIX E RSS PDL E-1

E1 Type DEFS and Global Data E-1
E2 Signal Generation Manager (SSGM) CSU E-47
E3 Signal Generation Controller (SGEN) CSU E-69
E4 Run Initialization (SRUN) CSU E-91
E5 Burst Initialization (SBIT) CSU E-139
E6 Burst Modelling (SBMT) CSU E-185
E7 RSS Utilities (SUTL) CSU E-243
E8 Playback Manager (SPMT) CSU E-257
E9 DSP Interface (TDIT) CSU E-275
E10 DSP Output (TIOP) CSU E-289
E11 RPC Server (TRSV) CSU E-303

APPENDIX E

RSS PDL

E1 Type DEFS and Global Data

THIS PAGE INTENTIONALLY LEFT BLANK

```
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
{}{}{}{}                                        {}{}{}{}
{}{}{}{}                                        {}{}{}{}
{}{}{}{}                                        {}{}{}{}
  {}{}                                            {}{}
  {}{}              CDADL PROCESSOR               {}{}
  {}{}                                            {}{}
  {}{}                                            {}{}
  {}{}               VERSION 2.4.6                {}{}
  {}{}                                            {}{}
  {}{}          SERIAL NUMBER: C36U0011           {}{}
  {}{}                                            {}{}
  {}{}              AUTHORIZED USER:              {}{}
  {}{}             Raytheon - Canada              {}{}
  {}{}                                            {}{}
  {}{}                                            {}{}
  {}{}                                            {}{}
  {}{}                                            {}{}
  {}{}                                            {}{}
{}{}{}{}                                        {}{}{}{}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
```

1 FILE sgen_priv.h
21 REPORT DATA DICTIONARY
35 REPORT Requirements Traceability
36 REPORT INVOCATION TREE
36 REPORT ERROR SUMMARY

E-3

```
/*============================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sgen_priv.h - RSS Signal Generation global header file

Responsible Organization:

Raytheon Canada Limited

Purpose:        This header file declares global variables and data structures
used within the RSS Signal Generation function.
It is formatted for processing by the CDADL processing
language.

Designer:       Marg Baskin

Change History:

SCR   Date         Author   Change made
---  -----------  ------  ---------------------------------------
n/a  08-APR-1993   mdb     Initial Release


============================================================================*/

#ifndef SGEN_PRIV_H
#define SGEN_PRIV_H
#include "spd_intf.h"
/****************************************************************************

defined constants and macros

****************************************************************************/

#define c_sgen_earth_radius  6378000
/*%
Radius of the earth (m)
*/
#define c_sgen_earth_rotation 7.29211515e-5
/*%
Earth rotation (radians/s)
*/
#define c_sgen_grav_constant  3.986008e14
/*%
Earth gravitational constant (m-3/s-2)
*/
#define c_sgen_sp_of_light 2.998e8
/*%
Speed of light (m/s)
*/
#define c_sgen_boltzmann 1.381e-23
/*%
Boltzman's constant
*/
#define c_sgen_collision_frequency 10e4
/*%
Collision frequency (radians/s) at max. collision alt
*/
#define c_sgen_max_collision_alt 134000
/*%
altitude of maximum collision frequency (m)
*/
#define c_sgen_scale_height 45000
/*%
scale height for collision frequency (m)
*/
#define c_sgen_k_l_t 1.011138e-7
/*%
constant used for distinguishing quasi-longitudinal from quasi-transverse propagation
*/
#define c_sgen_k_long 8.4823e-25
/*%
constant used in equation for Faraday rotation for quasi-longitudinal
propagation
*/
#define c_sgen_k_trans 6.3617e-34
/*%
constant used in equation for Faraday rotation for quasi-transverse
propagation
*/
#define c_sgen_k_gyro 2.7973e6
/*%
constant used for computing the gyromagnetic frequency
*/
#define c_sgen_k_ion_atten 1.167e-6
```

```
/*%
constant used for calculating ionospheric attenuation
*/
#define c_sgen_trop_angles 30
/*%
number of angle slots in the tropospheric refraction database
*/
#define c_sgen_trop_alts 20
/*%
number of altitude slots in the tropospheric refraction database
*/
#define c_sgen_max_samples 768000
/*%
maximum number of samples per channel per CPI
*/
#define c_sgen_nsegn 1024
/*%
noise segment length (in samples)
*/
#define c_sgen_noverlap 512
/*%
noise segment overlap (in samples)
*/
#define c_sgen_nsegoff c_sgen_noverlap/2
/*%
noise segment offset
*/
#define c_sgen_nsegj 1024
/*%
jammer segment length (in samples)
*/
#define c_sgen_joverlap 512
/*%
jammer segment overlap (in samples)
*/
#define c_sgen_jsegoff c_sgen_joverlap/2
/*%
jammer segment offset
*/
#define c_sgen_nsegif 1024
/*%
IF filter segment length
*/
#define c_sgen_overlapif c_sgen_nsegif/2
/*%
IF filter offset
*/
#define c_sgen_gauss_alpha 1.665
/*%
Gaussian alpha
*/
#define c_sgen_max_channels 32
/*%
maximum number of channels
*/
#define c_sgen_max_antennas 32
/*%
maximum number of antennas
*/
#define c_sgen_max_nsegin 1024
/*%
interpolation segment length
*/
#define c_sgen_ad_min -2e13
/*%
A/D minimum cut-off value
*/
#define c_sgen_ad_max 2e13-1
/*%
A/D maximum cut-off value
*/
#define c_sgen_max_rs 50000
/*%
Maximum number of range swaths
*/
#define c_sgen_max_iso 5000
/*%
Maximum number of isodops
*/
#define c_sgen_max_isoelements 5000
/*%
Maximum number of isodop elements
*/
#define c_sgen_max_orbit_positions 1000
/*%
Maximum number of elements in the orbital position sparse sequence
*/
#define c_sgen_max_npstot 10000
/*%
```

```
Maximum npstot value
*/
#define c_sgen_replica_max 10000
/*%
maximum length of pulse compression replica
*/
#define c_sgen_max_oversamples 32
/*%
maximum oversampling factor
*/
#define c_sgen_lat_lines 10801
/*%
number of lines of latitude for clutter map database
*/
#define c_sgen_min_arc PI/(180*60)
/*%
radians per minute of arc
*/
#define c_sgen_max_cpis c_spd_max_DI * c_spd_max_NCI * c_spd_max_CPI
/*%
maximum number of CPIs in a run
*/
#define c_sgen_ec_seed 100
/*%
earth clutter seed integer, for use when seed is RSS-generated
*/
#define c_sgen_discrete_seed 200
/*%
discrete clutter seed integer, for use when seed is RSS-generated
*/
#define c_sgen_noise_seed 73
/*%
noise seed integer, for use when seed is RSS-generated
*/
#define c_sgen_scintillation_seed 400
/*%
scintillation seed integer, for use when seed is RSS-generated
*/
#define c_sgen_vc_seed 300
/*%
volumetric clutter seed integer, for use when seed is RSS-generated
*/
#define c_sgen_target_seed(i) (26783+i*71)
/*%
target seed integer, for use when seed is RSS-generated
*/
#define c_sgen_jammer_seed(i) (1001+i*65)
/*%
jammer seed integer, for use when seed is RSS-generated
*/
#define c_sgen_sparse_ionos 200
/*%
maximum number of samples in the sparse ionospheric sequence
*/

/***************************************************************************

Control and status structures

***************************************************************************/

typedef enum
    {
    s_sgen_idle = 0,
    s_sgen_running=1,
    s_sgen_aborting=2,
    s_sgen_terminating=3
    } t_sgen_run_state;
/*%
signal generation internal running state.  This is separated out of RSS_State to allow
the tasks to maintain intermediate states "aborting" and "terminating"
*/

typedef enum
    {
    s_sgen_status_idle=0,
    s_sgen_srun_working=1,
    s_sgen_srun_terminated=2,
    s_sgen_srun_finished=3,
    s_sgen_sbit_working=4,
    s_sgen_sbit_terminated=5,
    s_sgen_sbit_finished=6,
    s_sgen_sbmt_working=7,
    s_sgen_sbmt_terminated=8,
    s_sgen_sbmt_finished=9
    } t_sgen_active_status;
/*%
Active state of the signal generation, used for state ordering and
orderly termination.  This status indicates which of the signal
```

E 6

```
generation child processes (srun, sbit, sbmt) is currently active,
as well as its operating state (working, terminated with error, finished)
*/

/**************************************************************************

Generic structure types

**************************************************************************/

typedef struct
    {
    t_spd_lat    latitude;
    t_spd_lon    longitude;
    t_spd_real   ssp_orbit_radius;
    } t_sgen_ecr_vector;
/*%
column vector of SBR position in ECR cartesians
*/

typedef struct
    {
    t_spd_real      ssp_orbit_radius;
    t_spd_lat       latitude;
    t_spd_lon       longitude;
    } t_sgen_sphere_pos_vect;
/*%
Column vector of position in spherical coordinates
*/

typedef struct
    {
    t_spd_real cart_x;
    t_spd_real cart_y;
    t_spd_real cart_z;
    } t_sgen_cart_vector;
/*%
column vector of position in cartesian coordinates
*/

typedef struct
    {
    t_spd_real range;
    t_spd_real elevation_angle;
    t_spd_real azimuth;
    } t_sgen_frd_sphere;
/*%
generic FRD spherical coordinates
*/

typedef t_spd_real t_sgen_rot_matrix[3][3];

/*%
general type used for an angular rotation matrix
*/

typedef t_spd_double t_sgen_elapse_time;
/*%
Time of day, relative to run start
*/

typedef t_spd_real t_sgen_seed;
/*%
seed parameter used for generating a random number
*/


/**************************************************************************

Run-level parameters  (These parameters are either fixed for
the whole run, or maintain overall consistancy for the whole run.)

**************************************************************************/

typedef t_spd_real t_sgen_orbit_radius;
/*%
This variable specifies the SBR orbit radius in kilometers
calculated from earth_radius + altitude.  Fixed per run.
*/

typedef t_spd_real t_sgen_rotational_velocity;
/*%
SBR rotational velocity in radians per second.  Fixed per run.
*/

typedef t_spd_real t_sgen_k1;
/*%
constant angle (1/orbit radius).  This variable has no abstract meaning,
but is used frequently throughout the algorithm
```

```
*/

typedef t_spd_real t_sgen_k3;
/*%
constant angle (earth's rotational velocity * cos(inclination) -
SBR rotational velocity).
This variable has no abstract meaning, but is used frequently
throughout the algorithm
*/

typedef t_spd_card8 t_sgen_replica_interpolation_factor;
/*%
replica interpolation factor.  Fixed per run.
*/

typedef struct
    {
    t_sgen_elapse_time      time_zero;
    t_spd_double            omega_zero;
    } t_sgen_orbital_model_coeffs;
/*%
Orbital model coefficients.  Fixed per run.
*/
/*%table
Members:
time_zero       t_sgen_elapse_time (time at which ascending node
crosses equatorial plane)
omega_zero      t_spd_double (longitude of ascending node at time zero)
*/

typedef t_spd_real t_sgen_trop_ref_db[c_sgen_trop_alts][c_sgen_trop_angles];
/*%
tropospheric refraction database.  Rows -> altitude; columns -> angles
each entry represents the refraction induced range error.  Fixed per run.
Size: 2400 bytes.
*/

typedef t_spd_card8 t_sgen_num_ant;
/*%
number of antennas in current run.  Fixed per run.
*/

typedef t_spd_card8 t_sgen_antenna_idx[c_sgen_max_channels];
/*%
antenna number for channel "x".  Fixed per run.
*/

typedef t_spd_card8 t_sgen_num_chan;
/*%
number of channels.  Fixed per run.
*/

typedef t_spd_complex8 t_sgen_noise_window[c_sgen_nsegn];
/*%
noise window parameters.  Fixed per run. Size: 8192 bytes.
*/

typedef t_spd_card8 t_sgen_first_channel[c_sgen_max_antennas];
/*%
array specifying the first channel number associated with each antenna.
Fixed per run. Size: 22 bytes.
*/

typedef t_spd_card16 t_sgen_out_off;
/*%
Offset of first sample to use for decimation.  Fixed per run.
*/

typedef t_spd_complex8
t_sgen_if_filter_response[c_sgen_max_antennas][c_sgen_nsegif];
/*%
IF filter response matrix.  Fixed per run. Size: 180,224 bytes.
*/

typedef t_spd_complex8 t_sgen_differentiator[c_sgen_nsegif];
/*%
frequency response of differentiator.  Fixed per run.
Size: 8192 bytes.
*/

typedef t_spd_real t_sgen_omega_if;
/*%
A/D IF angular frequency.  Fixed per run.
*/

typedef t_spd_complex8 t_sgen_iq_parameters[c_sgen_max_antennas];
/*%
I/Q parameters for antennas.  Fixed per run. Size: 176 bytes.
*/
```

```

typedef t_spd_complex8 t_sgen_ad_parameters[c_sgen_max_channels];
/*%
A/D parameters for channels.  Fixed per run. Size: 176 bytes.
*/

typedef t_spd_real t_sgen_range_quant;
/*%
range quantisation.  Fixed per run.
*/

typedef t_spd_card16 t_sgen_num_rs;
/*%
number of range-swaths.  Fixed per run.
*/

typedef t_spd_int8 t_sgen_elevation_angles[c_sgen_max_rs];
/*%
vector of elevation angles.  Fixed per run.  Size: 50,000 bytes.
*/

typedef t_spd_real t_sgen_velocity;
/*%
SBR velocity.  Fixed per run.
*/

typedef t_spd_real t_sgen_slant_range;
/*%
slant range to horizon. Fixed per run.
*/

typedef t_spd_real t_sgen_iratio;
/*%
ratio of noise/jammer oversampling factor to clutter oversampling
factor.  Fixed per run.
*/

typedef t_spd_complex8 t_sgen_iwindow[c_sgen_max_nsegin];
/*%
interpolation window vector.  Fixed per run. Size: 8192 bytes.
*/

typedef t_spd_real t_sgen_nonlinearity_params[c_sgen_max_antennas][4];
/*%
table of nonlinearity parameters for the antennas.  Fixed per run.
Size: 352 bytes.
*/

typedef t_spd_card16 t_sgen_number_of_cpis;
/*%
Total number of CPIs in the current run.  Fixed per run.
*/

typedef t_spd_card16 t_sgen_cpi_counter;
/*%
Current CPI.  (NOTE: this count is from the first CPI, not the start of
the current NCI or dwell.)
*/

typedef t_spd_card8 t_sgen_npplat[c_sgen_lat_lines];
/*%
vector representing number of clutter patches per latitude line.
Fixed per run.  Size: 18,801 bytes.
*/

typedef struct
    {
    t_spd_lon         lon_min;
    t_spd_lon         lon_max;
    t_spd_card8       patch_idx;
    } t_sgen_clutter_map_db [c_sgen_lat_lines][c_cend_max_patches];
/*%
clutter map database used for assigning clutter type to range-doppler
scatterer during generation.  Fixed per run.  Maximum number of
elemental structures is 1,080,100, giving a worst-case size of
18,361,700 bytes (17.5M).
*/
/*%table
Type              Member
t_spd_lat          latitude;
t_spd_lon          longitude;
t_spd_card8        clutter_type;
*/

typedef struct
    {
    t_sgen_seed     noise_seeds[97];
    t_sgen_seed     scintillation_seeds[97];
    t_sgen_seed     earth_clutter_seeds[97];
    t_sgen_seed    discrete_clutter_seeds[97];
    t_sgen_seed    volumetric_clutter_seeds[97];
    t_sgen_seed    jammer_seeds[c_cend_max_jammers][97];
    t_sgen_seed    target_seeds[c_cend_max_targets][97];
    } t_sgen_all_seeds;
/*%
seed vectors for all random number sequences used from CPI to CPI.
Each seed vector carries through from number generation to number
generation independently.  Size: 79,540 bytes.
*/

typedef struct
    {
    t_sgen_number_of_cpis                  number_of_cpis;
    t_sgen_orbit_radius                    orbit_radius;
    t_sgen_rotational_velocity             rotational_velocity;
    t_sgen_k1                              k1;
    t_sgen_k3                              k3;
    t_sgen_velocity                        velocity;
    t_sgen_range_quant                     range_quant;
    t_sgen_slant_range                     slant_range;
    t_sgen_orbital_model_coeffs            orbital_model_coeffs;
    t_sgen_num_ant                         number_of_antennas;
    t_sgen_num_chan                        number_of_channels;
    t_sgen_noise_window                    noise_window;
    t_sgen_first_channel                   first_channel;
    t_sgen_cut_off                         cut_off;
    t_sgen_if_filter_response              if_filter_response;
    t_sgen_differentiator                  differentiator;
    t_sgen_omega_if                        omega_if;
    t_sgen_iq_parameters                   iq_parameters;
    t_sgen_ad_parameters                   ad_parameters;
    t_sgen_num_rs                          num_rs;
    t_sgen_elevation_angles                elevation_angles;
    t_sgen_replica_interpolation_factor    replica_interpolation_factor;
    t_sgen_trop_ref_db                     trop_ref_db;
    t_sgen_antenna_idx                     antenna_idx;
    t_sgen_iratio                          iratio;
    t_sgen_iwindow                         iwindow;
    t_sgen_nonlinearity_params             nonlinearity_params;
    t_sgen_npplat                          npplat;
    t_sgen_clutter_map_db                  clutter_map_db;
    } t_sgen_run_constants;
/*%
computed constants used for the whole run
*/
/*%table
Members:                      Types:
number_of_cpis               t_sgen_number_of_cpis
orbit_radius                 t_sgen_orbit radius
rotational_velocity          t_sgen_rotational_velocity
k1                           t_sgen_k1
k3                           t_sgen_k3
velocity                     t_sgen_velocity
slant_range                  t_sgen_slant_range
orbital_model_coeffs         t_sgen_orbital_model_coeffs
number_of_antennas           t_sgen_num_ant
number_of_channels           t_sgen_num_chan
noise_window                 t_sgen_noise_window
first_channel                t_sgen_first_channel
cut_off                      t_sgen_cut_off
if_filter_response           t_sgen_if_filter_response
differentiator               t_sgen_differentiator
omega_if                     t_sgen_omega_if
iq_parameters                t_sgen_iq_parameters
ad_parameters                t_sgen_ad_parameters
range_quant                  t_sgen_range_quant
num_rs                       t_sgen_num_rs
elevation_angles             t_sgen_elevation_angles
replica_interpolation_factor  t_sgen_replica_interpolation_factor
trop_ref_db                  t_sgen_trop_ref_db
antenna_idx                  t_sgen_antenna_idx
iratio                       t_sgen_iradio
iwindow                      t_sgen_iwindow
nonlinearity_params          t_sgen_nonlinearity_params
npplat                       t_sgen_npplat
clutter_map_db               t_sgen_clutter_map_db
*/

typedef t_spd_card8 t_sgen_percent_complete;
/*%
Run percentage complete at time of last progress message transmission
*/

typedef t_spd_card8 t_sgen_current_dwell;
/*%
dwell counter
*/
typedef t_spd_card8 t_sgen_current_nci;
```

E-10

```
/*%
NCI counter
*/
typedef t_spd_card8 t_sgen_current_cpi;
/*%
CPI counter
*/

typedef struct
    {
    t_sgen_percent_complete    percent_complete;
    t_sgen_current_dwell       current_dwell;
    t_sgen_current_nci         current_nci;
    t_sgen_current_cpi         current_cpi;
    t_sgen_cpi_counter         cpi_counter;
    t_sgen_all_seeds           all_seeds;
    } t_sgen_r[illegible]arameters;
/*%
parameters which maintain their overall integrity across the course
of the run
*/
/*%table
Type                             Member
t_sgen_percent_complete            percent_complete
t_sgen_current_dwell               current_dwell
t_sgen_current_nci                 current_nci
t_sgen_current_cpi                 current_cpi
t_sgen_cpi_counter                 cpi_counter
t_sgen_all_seeds                   all_seeds
*/

/****************************************************************************

Dwell-level parameters
These are recalculated at the beginning of each new dwell

****************************************************************************/

typedef t_spd_lat t_sgen_beam_centre_latitude;
/*%
Beam centre latitude for the current dwell.
*/

typedef t_spd_lon t_sgen_beam_centre_longitude;
/*%
beam centre longitude for the current dwell.
*/

typedef struct
    {
    t_sgen_beam_centre_latitude     beam_centre_latitude;
    t_sgen_beam_centre_longitude    beam_centre_longitude;
    } t_sgen_dwell_database;
/*%
Working variables for the current dwell
*/
/*%table
Type                             Member
t_sgen_beam_centre_latitude           beam_centre_latitude
t_sgen_beam_centre_longitude          beam_centre_longitude
*/


/****************************************************************************

NCI level parameters
These are recalculated at the beginning of each new NCI

****************************************************************************/

typedef t_spd_real t_sgen_range_bins;
/*%
number of range bins in the current NCI,
*/

typedef t_spd_real t_sgen_coh_burst_time;
/*%
coherent burst time in seconds (pulses per cpi/PRF) for the current NCI
*/

typedef t_spd_real t_sgen_receive_ref_time;
/*%
receive reference time for the NCI (t_rx_ref)
*/

typedef struct
    {
    t_sgen_range_bins        range_bins;
    t_sgen_coh_burst_time    coh_burst_time;
    t_sgen_receive_ref_time  receive_ref_time;
    } t_sgen_nci_database;
/*%
Working variables for the current NCI, recalculated at the
beginning of each NCI
*/

/****************************************************************************

CPI level parameters

****************************************************************************/

typedef t_spd_real t_sgen_dop_quant;
/*%
doppler quantization
*/

typedef t_spd_real t_sgen_equi_range_vector[c_sgen_max_rs];
/*%
equi-range vector.  Max. size: 200,000 bytes
*/

typedef t_spd_card16 t_sgen_isoelements[c_sgen_max_iso];
/*%
vector containing the number of patches for each isodop.
Max. size: 10,000 bytes
*/

typedef t_spd_real t_sgen_radar_wavelength;
/*%
radar wavelength for the current CPI
*/

typedef t_spd_real t_sgen_sbr_heading;
/*%
instantaneous SBR heading
*/

typedef t_spd_sat_dir t_sgen_current_direction;
/*%
current satellite direction
*/

typedef t_spd_real t_sgen_k2;
/*%
constant (earth's rotational velocity * cos(SBR heading) *
cos(SSP latitude)
*/

typedef t_spd_real t_sgen_max_dop_angle;
/*%
angle of maximum doppler in radians
*/

typedef t_spd_card16 t_sgen_num_isodops;
/*%
number of isodops in range-doppler map
*/

typedef t_spd_real t_sgen_delta_rf;
/*%
change in RF between the previous and current CPIs
*/

typedef t_spd_location             t_sgen_SSP_current;
/*%
current SSP Latitude/longitude pair
*/

typedef struct
    {
    t_sgen_dop_quant              doppler_quant;
    t_sgen_equi_range_vector      equi_range_vector;
    t_sgen_isoelements            isoelements;
    t_sgen_radar_wavelength       radar_wavelength;
    t_sgen_sbr_heading            sbr_heading;
    t_sgen_k2                     k2;
    t_sgen_max_dop_angle          max_dop_angle;
    t_sgen_num_isodops            num_isodops;
    t_sgen_delta_rf               delta_rf;
    t_sgen_SSP_current            SSP_current;
    t_sgen_current_direction      current_direction;

    } t_sgen_cpi_database;
/*%
working variables for the current CPI, recalculated at the beginning
of each burst
*/
```

```
/*%table
Type                          Member
t_sgen_dop_quant                  doppler_quant
t_sgen_equi_range_vector          equi_range_vector
t_sgen_isoelements                isoelements
t_sgen_radar_wavelength           radar_wavelength
t_sgen_sbr_heading                sbr_heading
t_sgen_k2                         k2
t_sgen_max_dop_angle              max_dop_angle
t_sgen_num_isodops                num_isodops
t_sgen_delta_rf                   delta_rf
t_sgen_SSP_current                SSP_current
t_sgen_current_direction          current_direction
*/

/************************************************************************

waveform coefficients

************************************************************************/

typedef t_spd_complex8 t_sgen_pulse_compression_replica[c_sgen_replica_max];
/*%
pulse compression replica for the current dwell.  This is before
decimation for storage.  Max size: 80,000 bytes.
*/

typedef t_sgen_elapse_time t_sgen_trans_start_time;
/*%
transmit start time of current CPI
*/
typedef t_sgen_elapse_time t_sgen_trans_end_time;
/*%
transmit end time of current CPI
*/
typedef t_sgen_elapse_time t_sgen_receive_start_time_ref;
/*%
receive start time of current CPI
*/
typedef t_sgen_elapse_time t_sgen_receive_end_time_ref;
/*%
receive end time of current CPI
*/

typedef t_spd_card32 t_sgen_jammer_samples;
/*%
number of jammer samples for the current CPI at oversampled rate
*/

typedef t_spd_card32 t_sgen_clutter_samples;
/*%
number of clutter samples for the current CPI at oversampled rate
*/
typedef t_spd_card32 t_sgen_num_samples;
/*%
number of samples in the CPI
*/

typedef t_spd_real t_sgen_adjusted_prf;
/*%
pulse duration for the current dwell
*/
typedef struct
     {
     t_sgen_pulse_compression_replica    pulse_compression_replica;
     t_sgen_trans_start_time             trans_start_time;
     t_sgen_trans_end_time               trans_end_time;
     t_sgen_receive_start_time_ref       receive_start_time_ref;
     t_sgen_receive_end_time_ref         receive_end_time_ref;
     t_sgen_jammer_samples               jammer_samples;
     t_sgen_clutter_samples              clutter_samples;
     t_sgen_num_samples                  num_samples;
     t_sgen_adjusted_prf                 adjusted_prf;
     } t_sgen_waveform_coeffs;
/*%
Waveform coefficients
*/
/*%table
Type                               Member
t_sgen_pulse_compression_replica    pulse_compression_replica;
t_sgen_trans_start_time             trans_start_time;
t_sgen_trans_end_time               trans_end_time;
t_sgen_receive_start_time_ref       receive_start_time_ref;
t_sgen_receive_end_time_ref         receive_end_time_ref;
t_sgen_jammer_samples               jammer_samples;
t_sgen_clutter_samples              clutter_samples;
t_sgen_num_samples                  num_samples;
t_sgen_adjusted_prf                 adjusted_prf;
*/
```

```

/****************************************************************************

attenuation

****************************************************************************/

typedef t_spd_card16 t_sgen_nprecip;
/*%
number of precipitation cells in precipitation clutter database
*/

typedef struct
    {
    t_spd_real      cell_azimuth_extent_x;
    t_spd_real      cell_azimuth_extent_y;
    t_spd_real      cell_elevation_x;
    t_spd_real      cell_elevation_y;
    t_spd_real      cell_attenuation;
    t_spd_real      cell_thickness;
    } t_sgen_precip_dbase[c_cend_max_cells];
/*%
precipitation database generated per cpi. Max. size: 2400 bytes.
*/
/*%table
type            member
t_spd_real      cell_azimuth_extent_x;
t_spd_real      cell_azimuth_extent_y;
t_spd_real      cell_elevation_x;
t_spd_real      cell_elevation_y;
t_spd_real      cell_attenuation;
t_spd_real      cell_thickness;
*/

typedef t_spd_real t_sgen_at_attenuation[90];
/*%
table of atmospheric attenuation factors for the current CPI.
Size: 360 bytes.
*/

typedef struct
    {
    t_sgen_nprecip              nprecip;
    t_sgen_precip_dbase         precip_dbase;
    t_sgen_at_attenuation       at_attenuation;
    } t_sgen_attenuation_database;
/*%
attenuation database
*/
/*%table
Type                    Member
t_sgen_nprecip              nprecip
t_sgen_precip_dbase         precip_dbase
t_sgen_at_attenuation       at_attenuation
*/

/****************************************************************************

Range-doppler map & earth clutter database

****************************************************************************/

typedef t_spd_card16 t_sgen_num_iso_elements[c_sgen_max_isoelements];
/*%
vector indicating the number of patches in each isodop.
Max size: 10,000 bytes.
*/

typedef t_spd_card16 t_sgen_cpa[c_sgen_max_iso][c_sgen_max_isoelements];
/*%
Clutter patch area matrix.  In matrix form, the number of rows is
given by NumIso and the number of elements in row j is given by
num_iso_elements(j).  Because of the size (gigantic) and the irregular
shape, this matrix will be dynamically allocated row by row, and
referenced by pointer array only.  Above def. for clarity.
Max. size: 50,000,000 bytes.
*/

typedef t_spd_card16 t_sgen_cpc[c_sgen_max_iso][5];
/*%
clutter patch type matrix.  This matrix stores the number of patches
of each type (1 to 5) in each isodop.  Rows are isodops;
columns are patch type.  Max. size: 50,000 bytes.
*/

typedef t_spd_real t_sgen_graz[c_sgen_max_rs];
/*%
vector of grazing angles. Max size: 200,000 bytes.
*/
```

E-14

```

typedef t_spd_real t_sgen_range[c_sgen_max_re];
/*%
vector of mid-swath ranges.  Max size: 200,000 bytes.
*/

typedef struct
    {
    t_sgen_cpc    cpc;
    t_sgen_cpa    cpa;
    t_sgen_graz   graz;
    t_sgen_range  range;
    } t_sgen_ec_database;
/*%
Earth Clutter Database for current CPI
*/
/*%table
Type                    Member
t_sgen_cpc                   cpc
t_sgen_cpa                   cpa
t_sgen_graz                  graz
t_sgen_range                 range
*/

/*****************************************************************************

Volumetric Clutter database
(precipitation clutter database & chaff clutter database)

*****************************************************************************/

typedef t_spd_real t_sgen_frolloff[c_cend_max_cells];
/*%
precipitation cell spectral roll-off frequency.  Max size: 400 bytes.
*/

typedef t_spd_real t_sgen_fpeak[c_cend_max_cells];
/*%
precipitation cell peak frequency.    Max size: 400 bytes.
*/

typedef t_spd_real t_sgen_nps[c_cend_max_cells];
/*%
number of elements in precipitation cells.  Max size: 400 bytes.
*/

typedef t_spd_real t_sgen_atmos_atten[c_cend_max_cells];
/*%
atmospheric attenuation through precipitation cells.
Max size: 400 bytes.
*/

typedef t_spd_card16 t_sgen_npstot;
/*%
total number of elements in precipitations cells (summation of t_sgen_nps)
*/

typedef t_spd_real t_sgen_precip_rcs[c_sgen_max_npstot];
/*%
radar cross section of precipitation cells.  Max size: 40,000 bytes.
*/

typedef t_spd_real t_sgen_pscatloc[c_sgen_max_npstot][3];
/*%
precipitation scatterer locations. Max size: 120,000 bytes.
*/

typedef struct
    {
    t_sgen_nprecip         nprecip;
    t_sgen_frolloff        frolloff;
    t_sgen_fpeak           fpeak;
    t_sgen_nps             nps;
    t_sgen_atmos_atten     atmos_atten;
    t_sgen_npstot          npstot;
    t_sgen_precip_rcs      precip_rcs;
    t_sgen_pscatloc        pscatloc;
    } t_sgen_precip_db;
/*%
precipitation database for the current CPI, generated as part of
volumetric clutter database
*/
/*%table
Type                Member
t_sgen_nprecip           nprecip
t_sgen_frolloff          frolloff
t_sgen_fpeak             fpeak
t_sgen_nps               nps
t_sgen_atmos_atten       atmos_atten
```

```
t_sgen_npstot          npstot
t_sgen_precip_rcs      precip_rcs
t_sgen_pscatloc        pscatloc
*/

typedef t_spd_card16 t_sgen_nchaffs;
/*%
number of chaff cells in chaff clutter database
*/

typedef t_spd_real t_sgen_fcrolloff[c_cend_max_chaffs];
/*%
chaff cell spectral roll-off frequency.  Max size: 400 bytes.
*/

typedef t_spd_real t_sgen_fcpeak[c_cend_max_chaffs];
/*%
chaff cell peak frequency.  Max size: 400 bytes.
*/

typedef t_spd_real t_sgen_chaf_atmos_atten[c_cend_max_chaffs];
/*%
atmospheric attenuation through chaff cells.  Max size: 400 bytes.
*/

typedef t_spd_real t_sgen_chaff_rcs[c_cend_max_chaffs];
/*%
radar cross section of chaff cells.  Max size: 400 bytes.
*/

typedef t_spd_real t_sgen_chaff_loc[c_cend_max_chaffs][3];
/*%
chaff scatterer locations.  Max size: 400 bytes.
*/

typedef struct
    {
    t_sgen_nchaffs           nchaffs;
    t_sgen_fcrolloff         fcrolloff;
    t_sgen_fcpeak            fcpeak;
    t_sgen_chaf_atmos_atten  chaf_atmos_atten;
    t_sgen_chaff_rcs         chaff_rcs;
    t_sgen_chaff_loc         chaff_loc;
    } t_sgen_chaff_db;
/*%
chaff database for the current CPI, generated as part of volumetric
clutter database
*/
/*%table
Type                   Member
t_sgen_nchaffs           nchaff
t_sgen_fcrolloff         fcrolloff
t_sgen_fcpeak            fcpeak
t_sgen_chaf_atmos_atten  chaff_atmos_atten
t_sgen_chaff_rcs         chaff_rcs
t_sgen_chaff_loc         chaff_loc
*/

typedef struct
    {
    t_sgen_precip_db    precip_db;
    t_sgen_chaff_db     chaff_db;
    } t_sgen_vol_clutter_db;
/*%
Volumetric clutter database (precipitation database + chaff database)
*/

/****************************************************************************

Target database

****************************************************************************/

typedef t_spd_real t_sgen_target_range;
/*%
target range at start of burst
*/

typedef t_spd_real t_sgen_target_attenuation;
/*%
target atmospheric attenuation at start of burst
*/

typedef t_spd_boolean t_sgen_target_validity;
/*%
records whether or not an individual target is within the FoV for the
current CPI
*/
```

```
typedef t_spd_real t_sgen_gc_radius;
/*%
target great circle radius
*/

typedef t_spd_real t_sgen_ang_vel;
/*%
target angular velocity
*/

typedef t_spd_real t_sgen_t_rot_angle;
/*%
target rotation angle at current time
*/

typedef struct
     {
    t_sgen_target_validity      target_validity;
    t_sgen_target_range         target_range;
    t_sgen_target_attenuation   target_attenuation;
    } t_sgen_target_database[c_cend_max_targets];
/*%
target database.  Max. size: 1200 bytes.
*/
/*%table
Type                         Member
t_sgen_target_validity         target_validity
t_sgen_target_range            target_range
t_sgen_target_attenuation      target_attenuation
*/

/**************************************************************************

Jammer database

**************************************************************************/

typedef t_spd_real t_sgen_jammer_range;
/*%
jammer range at start of burst
*/

typedef t_spd_real t_sgen_jammer_attenuation;
/*%
jammer atmospheric attenuation at start of burst
*/

typedef t_spd_boolean t_sgen_jammer_validity;
/*%
records whether or not an individual jammer is within the FoV for
the current CPI
*/

typedef struct
     {
    t_sgen_jammer_validity      jammer_validity;
    t_sgen_jammer_range         jammer_range;
    t_sgen_jammer_attenuation   jammer_attenuation;
    } t_sgen_jammer_database[c_cend_max_jammers];
/*%
jammer database. Max. size: 1200 bytes.
*/
/*%table
Type                         Member
t_sgen_jammer_validity         jammer_validity
t_sgen_jammer_range            jammer_range
t_sgen_jammer_attenuation      jammer attenuation
*/


/**************************************************************************

Ionospheric model coefficients

**************************************************************************/

typedef t_spd_real t_sgen_faraday_rot;
/*%
one-way Faraday rotation
*/

typedef t_spd_real t_sgen_ion_atten;
/*%
total one-way ionospheric attenuation in dB
*/

typedef t_spd_card8      t_sgen_seq_length;
/*%
the length of the sparse ionospheric sequence for the current burst,
in samples
*/

typedef t_spd_real    t_sgen_dt_ion;
/*%
the sampling time interval for the current sparse ionospheric seq.
*/

typedef struct
    {
    t_sgen_faraday_rot     faraday_rot;
    t_sgen_ion_atten       ion_atten;
    t_sgen_seq_length      seq_length;
    t_sgen_dt_ion          dt_ion;
    t_spd_complex8         sparse_ionos[c_sgen_sparse_ionos];
    } t_sgen_ionosphere_model_coeffs;
/*%
Ionospheric model coefficients.  Max size: 1613 bytes.
*/
/*%table
Type                 Member
t_sgen_faraday_rot   faraday_rot
t_sgen_ion_atten     ion_atten
t_sgen_seq_length    seq_length;
t_sgen_dt_ion        dt_ion;
t_spd_complex8       sparse_ionos[c_sgen_sparse_ionos];
*/

/*****************************************************************************

Noise parameters

*****************************************************************************/

typedef t_spd_complex8
t_sgen_noise_covar[c_sgen_max_antennas][c_sgen_max_antennas];
/*%
noise covariance matrix.  Max size: 3872 bytes.
*/

typedef t_spd_complex8 t_sgen_uchol[c_sgen_max_antennas][c_sgen_max_antennas];
/*%
Cholesky factor matrix associated with the noise covariance matrix.
Max size: 3872 bytes.
*/

typedef t_spd_real t_sgen_prev_rf;
/*%
RF of previous CPI (RF[-1]=0)
*/

typedef struct
    {
    t_spd_real pc_aperture_x;
    t_spd_real pc_aperture_y;
    } t_sgen_pc_aperture [c_spd_max_DPCA_grps];
/*%
x and y dimensions for phase centers in each group.  Max size: 16 bytes.
*/

typedef struct
    {
    t_spd_real pc_loc_x;
    t_spd_real pc_loc_y;
    }:t_sgen_pc_loc [c_spd_max_tot_ph_ctrs];
/*%
x and y locations for each phase centre.  Max size: 48 bytes.
*/

typedef t_spd_real t_sgen_cfa_ne [c_spd_max_DPCA_grps];
/*%
number of elements in CFA regular arrays. Size: 8 bytes.
*/

typedef t_spd_real t_sgen_aux_ne [c_spd_max_DPCA_grps];
/*%
number of elements in aux. regular arrays.  Size: 8 bytes.
*/

typedef t_spd_real t_sgen_cfa_scan;
/*%
scanning phase factors for CFA regular array
*/

typedef t_spd_real t_sgen_aux_scan [c_spd_max_DPCA_grps];
/*%
scanning phase factors for aux. regular array.  Size: 8 bytes.
*/
```

E-18

```
typedef struct
    {
    t_spd_real abs_el_loc_x;
    t_spd_real abs_el_loc_y;
    } t_sgen_abs_el_loc [c_spd_max_aux][c_spd_max_elm_arbtr_arr][c_sgen_max_antennas];
/*%
absolute location of each element of arbitrary array.
Max size: 45,056 bytes.
*/

typedef t_spd_complex8 t_sgen_el_wt[c_spd_max_aux][c_spd_max_elm_arbtr_arr][c_sgen_max_      &
                                                                                 antennas];
/*%
complex weight for each element of arbitrary array
Max size: 45,056 bytes.
*/

typedef t_spd_complex8 t_sgen_cor[c_sgen_max_antennas][c_sgen_max_antennas];
/*%
correlation coefficient between antennas.  Size: 3872 bytes.
*/

typedef t_spd_real t_sgen_ex_temp;
/*%
external noise temperature
*/

typedef t_spd_real t_sgen_noise_power[c_sgen_max_antennas];
/*%
noise power at each antenna.  Size: 88 bytes.
*/

typedef struct
    {
    t_sgen_noise_covar  noise_covar;
    t_sgen_uchol        uchol;
    t_sgen_prev_rf      prev_rf;
    } t_sgen_noise_coeffs;
/*%
Noise coefficients
*/
/*%table
Type                Member
t_sgen_noise_covar  noise_covar;
t_sgen_uchol        uchol;
t_sgen_prev_rf      prev_rf;
*/

/**************************************************************************

Antenna patterns

**************************************************************************/

typedef t_spd_complex8 t_sgen_tx_CPA_antenna_pattern[4][2048];
/*%
transmit antenna pattern.  This matrix represents four generated look-up
tables (GainU, GainV, EgainU, EgainV) of differing length, and will
be revised.  Here for sizing only.  Max size: 65,536 bytes.
*/

typedef t_spd_complex8 t_sgen_gainu[2048];
/*%
The principle axes array factor pattern, direction u
*/

typedef t_spd_complex8 t_sgen_gainv[2048];
/*%
The principle axes array factor pattern, direction v
*/

typedef t_spd_complex8 t_sgen_egainu[2048];
/*%
The principle axes elemental pattern, direction u
*/

typedef t_spd_complex8 t_sgen_egainv[2048];
/*%
The principle axes elemental pattern, direction v
*/

typedef t_spd_complex8 t_sgen_CPA_rx_antenna_pattern[4][2048];
/*%
receive antenna pattern.  One per receive antenna.
This matrix represents four generated look-up
tables (GainU, GainV, EgainU, EgainV) of differing length, and will
be revised.  Here for sizing only.  Max size: 65,536 bytes.
*/
```

```
typedef t_spd_complex8 t_sgen_RRA_tx_antenna_pattern[3][16284];
/*%
RRA transmit antenna pattern.  This matrix represents 3 subtables of
uneven length and will be revised.  Here for sizing only.
Max size: 393216 bytes.
*/

typedef t_spd_complex8 t_sgen_RRA_rx_antenna_pattern[3][16284];
/*%
RRA receive antenna pattern.  This matrix represents 3 subtables of
uneven length and will be revised.  Here for sizing only.
Max size: 393216 bytes.
*/

/************************************************************************

Burst database

************************************************************************/

typedef struct
    {
    t_sgen_ecr_vector      ecr_position;
    t_spd_real            sin_heading;
    t_spd_real            cos_heading;
    t_spd_real            sin_lat;
    t_spd_real            cos_lat;
    t_spd_real            sin_lon;
    t_spd_real            cos_lon;
    } t_sgen_sbr_orbit_position[c_sgen_max_orbit_positions];
/*%
sparce sequence of SBR positions during the current CPI.
Max size: 36,000 bytes.
*/
/*%table
TYPE                 MEMBER
t_sgen_ecr_vector       ecr_position; position in ECR caresians
t_spd_real            sin_heading;  sin of the instantaneous heading
t_spd_real            cos_heading;  cos of the instantaneous heading
t_spd_real            sin_lat;      sin of the SSP latitude
t_spd_real            cos_lat;      cos of the SSP latitude
t_spd_real            sin_lon;      sin of the SSP longitude
t_spd_real            cos_lon;      cos of the SSP longitude
*/
typedef struct
    {
    t_sgen_dwell_database             dwell_database;
    t_sgen_nci_database               nci_database;
    t_sgen_cpi_database               cpi_database;
    t_sgen_waveform_coeffs            waveform_coeffs;
    t_sgen_attenuation_database       attenuation_database;
    t_sgen_ec_database                earth_clutter_database;
    t_sgen_vol_clutter_db             vol_clutter_db;
    t_sgen_target_database            target_database;
    t_sgen_jammer_database            jammer_database;
    t_sgen_ionosphere_model_coeffs    ionosphere_model_coeffs;
    t_sgen_noise_coeffs               noise_coeffs;
    t_sgen_sbr_orbit_position         orbit_position;
    } t_sgen_burst_database;

/*%
initialized database for the current CPI, generated during
burst initialization
*/
/*%table
Type                              Member
t_sgen_dwell_database                dwell_database
t_sgen_nci_database                  nci_database
t_sgen_cpi_database                  cpi_database
t_sgen_waveform_coeffs               waveform_coeffs;
t_sgen_attenuation_database          attenuation_database;
t_sgen_earth_clutter_database        earth_clutter_database;
t_sgen_vol_clutter_db                vol_clutter_db;
t_sgen_target_database               target_database;
t_sgen_jammer_database               jammer_database;
t_sgen_ionosphere_model_coeffs       ionosphere_model_coeffs;
t_sgen_noise_coeffs                  noise_coeffs;
*/

/************************************************************************

Earth clutter modelling

************************************************************************/

typedef t_spd_complex8 t_sgen_earth_clutter_samples[c_sgen_max_samples][c_sgen_max_        &
                                                                              channels];
/*%
earth clutter intermediate product sample matrix for a single CPI
```

F-20

```
Max size: 135,168,000 bytes (roughly 129MB).
*/

/****************************************************************************

Interpolation

****************************************************************************/

typedef t_spd_card16 t_sgen_cutoff;
/*%
first element to use from the interpolated segment
*/

typedef t_spd_real t_sgen_nsegin;
/*%
input segment length for the interpolation routine
*/

typedef t_spd_complex8
t_sgen_oversamples [c_sgen_max_samples*c_sgen_max_oversamples][c_sgen_max_channels];
/*%
oversampled temporary matrix.  Max size: 540,672,000 bytes (roughly 516MB)
*/

/****************************************************************************

jammer modelling

****************************************************************************/

typedef struct
     {
     t_spd_real jam_x;
     t_spd_real jam_y;
     t_spd_real jam_z;
     } t_sgen_jammer_position;
/*%
Jammer position in frd coordinates

typedef struct
{
t_sgen_jammer_position jam_pos_beg;
t_sgen_jammer_position jam_pos_middle;
t_sgen_jammer_position jam_pos_end;
} t_sgen_jam_frd[c_cend_max_jammers];
/*%
position of jammers at beginning, middle, end of the time sequence.
Max size: 3600 bytes.
*/

typedef t_spd_complex8 t_sgen_jammer_window[c_sgen_nsegj];
/*%
jammer window vector (windowj).  Size: 8192 bytes.
*/

typedef t_spd_complex8 t_sgen_sweep_mod;
/*%
Jammer sweep modifier
*/

typedef t_spd_real t_sgen_seg_int;
/*%
Jammer segment interval (seg_int)
*/

typedef t_spd_real t_sgen_seg_time;
/*%
Jammer segment time (seg_time);
*/

typedef t_spd_card16 t_sgen_jstart_time;
/*%
Jammer active time segment start (tstart)
*/

typedef t_spd_card16 t_sgen_jstop_time;
/*%
Jammer active time segment stop (tstop)
*/


/****************************************************************************

Noise modelling

****************************************************************************/

typedef t_spd_real t_sgen_delta_f_noise;
```

E-2

```
/*%
frequency spacing used in noise generation
= (sampling rate * oversampling factor) / noise segment length
*/

typedef t_spd_complex8 t_sgen_spec[c_sgen_nsegn];
/*%
noise generation frequency-domain vector.  Size 8192 bytes.
*/

typedef t_spd_complex8 t_sgen_seg[c_sgen_nsegn];
/*%
noise generation subsegment vector.  Size 8192 bytes.
*/

typedef t_spd_complex8 t_sgen_chan_spec[c_sgen_nsegn];
/*%
post-IFFT noise generation subsegment frequency-domain vector.
Size: 8192 bytes.
*/

typedef t_spd_complex8 t_sgen_chan_seg[c_sgen_nsegn];
/*%
post-IFFT noise generation subsegment vector.  Size: 8192 bytes.
*/

/************************************************************************

aggregate returns

************************************************************************/

typedef t_spd_complex8
t_sgen_aggregate_oversamples[c_sgen_max_samples*c_sgen_max_oversamples][c_sgen_max_     &
                                                                            channels];

/*%
oversampled aggregate matrix.  Max size: 540,672,000 bytes (roughly 516MB)
*/

typedef t_spd_complex8
t_sgen_srsd_samples [c_sgen_max_samples][c_sgen_max_channels];
/*%
simulated radar signal data for one CPI (final, no oversampling,
but still floating point).  Max size: 135,168,000 bytes (roughly 129MB).
*/

typedef struct
    {
    t_spd_card32 number_of_cpis;
    t_sgen_adjusted_prf prf;
    t_sgen_num_samples number_of_samples[c_spd_max_CPI];
    } t_sgen_nci_srsd;
/*%
This structure defines information stored in the SRSD output file header
block for each NCI in the run.  Max size: 264 bytes.
*/
/*%table
TYPE                          MEMBER
t_spd_card32                    number_of_cpis
t_sgen_adjusted_prf             prf;
t_sgen_num_samples              number_of_samples[c_spd_max_CPI];
*/

typedef struct
    {
    char satdef pathname[c_spd_path_size];
    char envdef_pathname[c_spd_path_size];
    t_sgen_num_chan number_of_channels;
    t_spd_int32     number_of_ncis;
    t_sgen_nci_srsd nci_info[c_spd_max_NCI*c_spd_max_DI];
    } t_sgen_srsd_header;
/*%
this structure defines the header for the SRSD output file,
providing information needed to playback the generated data
*/
/*%table
TYPE                           MEMBER
char                         satdef_pathname[c_spd_path_size];
char                         envdef_pathname[c_spd_path_size];
t_sgen_num_chan              number_of_channels;
t_sgen_number_of_cpis        number_of_cpis;
t_sgen_cpi_srsd              cpi_info[c_sgen_max_cpis];
*/

/************************************************************************

front-end modelling

************************************************************************/
```

```

typedef t_egen_elapse_time t_egen_first_sample[c_egen_max_antennas];
/*%
time for first sample for each antenne
*/

/******************************************************************************

SRSD output file structures

******************************************************************************/

typedef short int t_egen_srsd_int;

/*%
The base type for stored radar samples
*/

typedef struct
    {
    t_egen_srsd_int in_phase;
    t_egen_srsd_int quadrature;
    } t_egen_srsd_sample;
/*%
Definition for one single complex radar sample, as output to the SRSD
file and the A/D intermediate product file
*/
typedef t_egen_srsd_sample
t_egen_cpi_data [c_egen_max_samples][c_spd_max_channels];

/*%
This block defines a full CPIs worth of radar data in the format to be
stored in the SRSD file and the A/D intermediate product file.
Max size: 67584000 bytes (roughly 64MB).
*/

#endif




```

```
==========================================================================================

SGEN_PRIV_H
    ** No Dictionary Definition Given **
        Macro << Global  PAGE 1  LINE 31 >>

c_sgen_ad_max
    A/D maximum cut-off value
        Macro << Global  PAGE 2  LINE 157 >>

c_sgen_ad_min
    A/D minimum cut-off value
        Macro << Global  PAGE 2  LINE 153 >>

c_sgen_boltzmann
    Boltzman's constant
        Macro << Global  PAGE 1  LINE 55 >>

c_sgen_collision_frequency
    Collision frequency (radians/s) at max. collision alt
        Macro << Global  PAGE 1  LINE 59 >>

c_sgen_discrete_seed
    discrete clutter seed integer, for use when seed is RSS-generated
        Macro << Global  PAGE 3  LINE 205 >>

c_sgen_earth_radius
    Radius of the earth (m)
        Macro << Global  PAGE 1  LINE 39 >>

c_sgen_earth_rotation
    Earth rotation (radians/s)
        Macro << Global  PAGE 1  LINE 43 >>

c_sgen_ec_seed
    earth clutter seed integer, for use when seed is RSS-generated
        Macro << Global  PAGE 3  LINE 201 >>

c_sgen_gauss_alpha
    Gaussian alpha
        Macro << Global  PAGE 2  LINE 137 >>

c_sgen_grav_constant
    Earth gravitational constant (m-3/s-2)
        Macro << Global  PAGE 1  LINE 47 >>

c_sgen_jammer_seed(i)
    jammer seed integer, for use when seed is RSS-generated
        Macro << Global  PAGE 3  LINE 225 >>

c_sgen_joverlap
    jammer segment overlap (in samples)
        Macro << Global  PAGE 2  LINE 121 >>

c_sgen_jsegoff
    jammer segment offset
        Macro << Global  PAGE 2  LINE 125 >>

c_sgen_k_gyro
    constant used for computing the gyromagnetic frequency
        Macro << Global  PAGE 1  LINE 85 >>

c_sgen_k_ion_atten
    constant used for calculating ionospheric attenuation
        Macro << Global  PAGE 1  LINE 89 >>

c_sgen_k_l_t
    constant used for distinguishing quasi-longitudinal from quasi-transverse propagation
        Macro << Global  PAGE 1  LINE 71 >>

c_sgen_k_long
    constant used in equation for Faraday rotation for quasi-longitudinal propagation
        Macro << Global  PAGE 1  LINE 75 >>

c_sgen_k_trans
    constant used in equation for Faraday rotation for quasi-transverse propagation
        Macro << Global  PAGE 1  LINE 80 >>

c_sgen_lat_lines
    number of lines of latitude for clutter map database
        Macro << Global  PAGE 3  LINE 189 >>

c_sgen_max_antennas
    maximum number of antennas
        Macro << Global  PAGE 2  LINE 145 >>
```

```
c_sgen_max_channels
    maximum number of channels
         Macro << Global  PAGE 2  LINE 141 >>

c_sgen_max_collision_alt
    altitude of maximum collision frequency (m)
         Macro << Global  PAGE 1  LINE 63 >>

c_sgen_max_cpis
    maximum number of CPIs in a run
         Macro << Global  PAGE 3  LINE 197 >>

c_sgen_max_iso
    Maximum number of isodops
         Macro << Global  PAGE 2  LINE 165 >>

c_sgen_max_isoelements
    Maximum number of isodop elements
         Macro << Global  PAGE 2  LINE 169 >>

c_sgen_max_npstot
    Maximum npstot value
         Macro << Global  PAGE 2  LINE 177 >>

c_sgen_max_nsegin
    interpolation segment length
         Macro << Global  PAGE 2  LINE 149 >>

c_sgen_max_orbit_positions
    Maximum number of elements in the orbital position sparse sequence
         Macro << Global  PAGE 2  LINE 173 >>

c_sgen_max_oversamples
    maximum oversampling factor
         Macro << Global  PAGE 3  LINE 185 >>

c_sgen_max_rs
    Maximum number of range swaths
         Macro << Global  PAGE 2  LINE 161 >>

c_sgen_max_samples
    maximum number of samples per channel per CPI
         Macro << Global  PAGE 2  LINE 101 >>

c_sgen_min_arc
    radians per minute of arc
         Macro << Global  PAGE 3  LINE 193 >>

c_sgen_noise_seed
    noise seed integer, for use when seed is RSS-generated
         Macro << Global  PAGE 3  LINE 209 >>

c_sgen_noverlap
    noise segment overlap (in samples)
         Macro << Global  PAGE 2  LINE 109 >>

c_sgen_nsegif
    IF filter segment length
         Macro << Global  PAGE 2  LINE 129 >>

c_sgen_nsegj
    jammer segment length (in samples)
         Macro << Global  PAGE 2  LINE 117 >>

c_sgen_nsegn
    noise segment length (in samples)
         Macro << Global  PAGE 2  LINE 105 >>

c_sgen_nsegoff
    noise segment offset
         Macro << Global  PAGE 2  LINE 113 >>

c_sgen_overlapif
    IF filter offset
         Macro << Global  PAGE 2  LINE 133 >>

c_sgen_replica_max
    maximum length of pulse compression replica
         Macro << Global  PAGE 3  LINE 181 >>

c_sgen_scale_height
    scale height for collision frequency (m)
         Macro << Global  PAGE 1  LINE 57 >>
```

```
                                DATA DICTIONARY
================================================================================

c_sgen_scintillation_seed
    scintillation seed integer, for use when seed is RSS-generated
        Macro << Global  PAGE 3  LINE 213 >>

c_sgen_sp_of_light
    Speed of light (m/s)
        Macro << Global  PAGE 1  LINE 51 >>

c_sgen_sparse_ionos
    maximum number of samples in the sparse ionospheric sequence
        Macro << Global  PAGE 3  LINE 229 >>

c_sgen_target_seed(i)
    target seed integer, for use when seed is RSS-generated
        Macro << Global  PAGE 3  LINE 221 >>

c_sgen_trop_alts
    number of altitude slots in the tropospheric refraction database
        Macro << Global  PAGE 2  LINE 97 >>

c_sgen_trop_angles
    number of angle slots in the tropospheric refraction database
        Macro << Global  PAGE 2  LINE 93 >>

c_sgen_vc_seed
    volumetric clutter seed integer, for use when seed is RSS-generated
        Macro << Global  PAGE 3  LINE 217 >>

t_sgen_CPA_rx_antenna_pattern
    receive antenna pattern.  One per receive antenna. This matrix represents four generated
    look-up tables (GainU, GainV, EgainU, EgainV) of differing length, and will be revised.
    Here for sizing only.  Max size: 65,536 bytes.
        Type << Global  PAGE 16  LINE 1416 >>

t_sgen_RRA_rx_antenna_pattern
    RRA receive antenna pattern.  This matrix represents 3 subtables of uneven length and will
    be revised.  Here for sizing only. Max size: 393216 bytes.
        Type << Global  PAGE 17  LINE 1431 >>

t_sgen_RRA_tx_antenna_pattern
    RRA transmit antenna pattern.  This matrix represents 3 subtables of uneven length and will
    be revised.  Here for sizing only. Max size: 393216 bytes.
        Type << Global  PAGE 17  LINE 1424 >>

t_sgen_SSP_current
    current SSP Latitude/longitude pair
        Type << Global  PAGE 9  LINE 778 >>

t_sgen_abs_el_loc
    absolute location of each element of arbitrary array. Max size: 45,056 bytes.
        Type << Global  PAGE 16  LINE 1360 >>

t_sgen_active_status
    Active state of the signal generation, used for state ordering and orderly termination.
    This status indicates which of the signal generation child processes (srun, sbit, sbmt) is
    currently active, as well as its operating state (working, terminated with error, finished)
        Type << Global  PAGE 3  LINE 264 >>

t_sgen_ad_parameters
    A/D parameters for channels.  Fixed per run. Size: 176 bytes.
        Type << Global  PAGE 6  LINE 447 >>

t_sgen_adjusted_prf
    pulse duration for the current dwell
        Type << Global  PAGE 10  LINE 860 >>

t_sgen_aggregate_oversamples
    oversampled aggregate matrix.  Max size: 540,672,000 bytes (roughly 516MB)
        Type << Global  PAGE 19  LINE 1634 >>

t_sgen_all_seeds
    seed vectors for all random number sequences used from CPI to CPI. Each seed vector carries
    through from number generation to number generation independently.  Size: 79,560 bytes.
        Type << Global  PAGE 7  LINE 539 >>

t_sgen_ang_vel
    target angular velocity
        Type << Global  PAGE 14  LINE 1163 >>

t_sgen_antenna_idx
    antenna number for channel "x".  Fixed per run.
        Type << Global  PAGE 5  LINE 399 >>

t_sgen_at_attenuation
    table of atmospheric attenuation factors for the current CPI. Size: 360 bytes.
        Type << Global  PAGE 11  LINE 925 >>
```

```
==================================================================================================

t_sgen_atmos_atten
    atmospheric attenuation through precipitation cells. Max size: 400 bytes.
        Type << Global  PAGE 12  LINE 1026 >>

t_sgen_attenuation_database
    attenuation database
              Type                    Member
           t_sgen_nprecip              nprecip
           t_sgen_precip_dbase         precip_dbase
           t_sgen_at_attenuation       at_attenuation

        Type << Global  PAGE 11  LINE 936 >>

t_sgen_aux_ne
    number of elements in aux. regular arrays.  Size: 8 bytes.
        Type << Global  PAGE 15  LINE 1321 >>

t_sgen_aux_scan
    scanning phase factors for aux. regular array.  Size: 8 bytes.
        Type << Global  PAGE 15  LINE 1331 >>

t_sgen_beam_centre_latitude
    Beam centre latitude for the current dwell.
        Type << Global  PAGE 8  LINE 662 >>

t_sgen_beam_centre_longitude
    beam centre longitude for the current dwell.
        Type << Global  PAGE 8  LINE 667 >>

t_sgen_burst_database
    initialized database for the current CPI, generated during burst initialization
              Type                           Member
         t_sgen_dwell_database            dwell_database
         t_sgen_nci_database              nci_database
         t_sgen_cpi_database              cpi_database
         t_sgen_waveform_coeffs           waveform_coeffs;
         t_sgen_attenuation_database      attenuation_database;
         t_sgen_earth_clutter_database    earth_clutter_database;
         t_sgen_vol_clutter_db            vol_clutter_db;
         t_sgen_target_database           target_database;
         t_sgen_jammer_database           jammer_database;
         t_sgen_ionosphere_model_coeffs   ionosphere_model_coeffs;
         t_sgen_noise_coeffs              noise_coeffs;

        Type << Global  PAGE 17  LINE 1482 >>

t_sgen_cart_vector
    column vector of position in cartesian coordinates
        Type << Global  PAGE 4  LINE 303 >>

t_sgen_cfa_ne
    number of elements in CFA regular arrays. Size: 8 bytes.
        Type << Global  PAGE 15  LINE 1316 >>

t_sgen_cfa_scan
    scanning phase factors for CFA regular array
        Type << Global  PAGE 15  LINE 1326 >>

t_sgen_chaff_atmos_atten
    atmospheric attenuation through chaff cells.  Max size: 400 bytes.
        Type << Global  PAGE 13  LINE 1089 >>

t_sgen_chaff_db
    chaff database for the current CPI, generated as part of volumetric clutter database
              Type                    Member
          t_sgen_nchaffs              nchaff
          t_sgen_fcrolloff            fcrolloff
          t_sgen_fcpeak               fcpeak
          t_sgen_chaf_atmos_atten     chaff_atmos_atten
          t_sgen_chaff_rcs            chaff_rcs
          t_sgen_chaff_loc            chaff_loc

        Type << Global  PAGE 13  LINE 1112 >>

t_sgen_chaff_loc
    chaff scatterer locations.  Max size: 400 bytes.
        Type << Global  PAGE 13  LINE 1099 >>

t_sgen_chaff_rcs
    radar cross section of chaff cells.  Max size: 400 bytes.
        Type << Global  PAGE 13  LINE 1094 >>

t_sgen_chan_seg
    post-IFFT noise generation subsegment vector.  Size: 8192 bytes.
        Type << Global  PAGE 19  LINE 1622 >>
```

```
==============================================================================================

t_sgen_chan_spec
    post-IFFT noise generation subsegment frequency-domain vector. Size: 8192 bytes.
        Type << Global  PAGE 19  LINE 1616 >>

t_sgen_clutter_map_db
    clutter map database used for assigning clutter type to range-doppler scatterer during
    generation.  Fixed per run.  Maximum number of elemental structures is 1,080,100, giving a
    worst-case size of 18,361,700 bytes (17.5M).
            Type               Member
          t_spd_lat          latitude;
          t_spd_lon          longitude;
          t_spd_card8        clutter_type;

        Type << Global  PAGE 6  LINE 516 >>

t_sgen_clutter_samples
    number of clutter samples for the current CPI at oversampled rate
        Type << Global  PAGE 10  LINE 851 >>

t_sgen_coh_burst_time
    coherent burst time in seconds (pulses per cpi/PRF) for the current MCI
        Type << Global  PAGE 8  LINE 699 >>

t_sgen_cor
    correlation coefficient between antennas.  Size: 3872 bytes.
        Type << Global  PAGE 16  LINE 1352 >>

t_sgen_cpa
    Clutter patch area matrix.  In matrix form, the number of rows is given by NumIso and the
    number of elements in row j is given by num_iso_elements(j).  Because of the size (gigantic)
    and the irregular shape, this matrix will be dynamically allocated row by row, and
    referenced by pointer array only.  Above def. for clarity. Max. size: 50,000,000 bytes.
        Type << Global  PAGE 11  LINE 959 >>

t_sgen_cpc
    clutter patch type matrix.  This matrix stores the number of patches of each type (1 to 5)
    in each isodop.  Rows are isodops; columns are patch type.  Max. size: 50,000 bytes.
        Type << Global  PAGE 11  LINE 969 >>

t_sgen_cpi_counter
    Current CPI.  (NOTE: this count is from the first CPI, not the start of the current MCI or
    dwell.)
        Type << Global  PAGE 6  LINE 499 >>

t_sgen_cpi_data
    This block defines a full CPIs worth of radar data in the format to be stored in the SRSD
    file and the A/D intermediate product file. Max size: 67584000 bytes (roughly 64MB).
        Type << Global  PAGE 20  LINE 1717 >>

t_sgen_cpi_database
    working variables for the current CPI, recalculated at the beginning of each burst
            Type                        Member
        t_sgen_dop_quant             doppler_quant
        t_sgen_equi_range_vector     equi_range_vector
        t_sgen_isoelements           isoelements
        t_sgen_radar_wavelength      radar_wavelength
        t_sgen_sbr_heading           sbr_heading
        t_sgen_k2                    k2
        t_sgen_max_dop_angle         max_dop_angle
        t_sgen_num_isodops           num_isodops
        t_sgen_delta_rf              delta_rf
        t_sgen_SSP_current           SSP_current
        t_sgen_current_direction     current_direction

        Type << Global  PAGE 9  LINE 737 >>

t_sgen_current_cpi
    CPI counter
        Type << Global  PAGE 8  LINE 627 >>

t_sgen_current_direction
    current satellite direction
        Type << Global  PAGE 9  LINE 752 >>

t_sgen_current_dwell
    dwell counter
        Type << Global  PAGE 7  LINE 619 >>

t_sgen_current_mci
    MCI counter
        Type << Global  PAGE 7  LINE 623 >>

t_sgen_delta_f_noise
    frequency spacing used in noise generation = (sampling rate * oversampling factor) / noise
    segment length
        Type << Global  PAGE 18  LINE 1600 >>
```

```
==========================================================================================

t_sgen_delta_rf
    change in RF between the previous and current CPIs
        Type << Global  PAGE 9  LINE 773 >>

t_sgen_differentiator
    frequency response of differentiator.  Fixed per run. Size: 8192 bytes.
        Type << Global  PAGE 5  LINE 431 >>

t_sgen_dop_quant
    doppler quantization
        Type << Global  PAGE 9  LINE 726 >>

t_sgen_dt_ion
    the sampling time interval for the current sparse ionospheric seq.
        Type << Global  PAGE 15  LINE 1250 >>

t_sgen_dwell_database
    Working variables for the current dwell
             Type                             Member
       t_sgen_beam_centre_latitude         beam_centre_latitude
       t_sgen_beam_centre_longitude        beam_centre_longitude

        Type << Global  PAGE 8  LINE 676 >>

t_sgen_earth_clutter_samples
    earth clutter intermediate product sample matrix for a single CPI Max size: 135,168,000
    bytes (roughly 129MB).
        Type << Global  PAGE 17  LINE 1509 >>

t_sgen_ec_database
    Earth Clutter Database for current CPI
            Type                 Member
             t_sgen_cpc            cpc
             t_sgen_cpa            cpa
             t_sgen_graz           graz
             t_sgen_range          range

        Type << Global  PAGE 13  LINE 992 >>

t_sgen_ecr_vector
    column vector of SBR position in ECR cartesians
        Type << Global  PAGE 4  LINE 283 >>

t_sgen_egainu
    The principle axes elemental pattern, direction u
        Type << Global  PAGE 16  LINE 1406 >>

t_sgen_egainv
    The principle axes elemental pattern, direction v
        Type << Global  PAGE 16  LINE 1411 >>

t_sgen_el_wt
    complex weight for each element of arbitrary array Max size: 45,056 bytes.
        Type << Global  PAGE 16  LINE 1346 >>

t_sgen_elapse_time
    Time of day, relative to run start
        Type << Global  PAGE 4  LINE 324 >>

t_sgen_elevation_angles
    vector of elevation angles.  Fixed per run.  Size: 50,000 bytes.
        Type << Global  PAGE 6  LINE 462 >>

t_sgen_equi_range_vector
    equi-range vector.  Max. size: 200,000 bytes
        Type << Global  PAGE 9  LINE 731 >>

t_sgen_ex_temp
    external noise temperature
        Type << Global  PAGE 16  LINE 1357 >>

t_sgen_faraday_rot
    one-way Faraday rotation
        Type << Global  PAGE 14  LINE 1234 >>

t_sgen_fcpeak
    chaff cell peak frequency.  Max size: 400 bytes.
        Type << Global  PAGE 13  LINE 1084 >>

t_sgen_fcrolloff
    chaff cell spectral roll-off frequency.  Max size: 400 bytes.
        Type << Global  PAGE 13  LINE 1079 >>

t_sgen_first_channel
    array specifying the first channel number associated with each antenna. Fixed per run. Size:
    22 bytes.
```

```
==========================================================================================================
t_sgen_first_channel  (continued)
        Type << Global  PAGE 5  LINE 414 >>

t_sgen_first_sample
    time for first sample for each antenna
        Type << Global  PAGE 20  LINE 1690 >>

t_sgen_fpeak
    precipitation cell peak frequency.    Max size: 400 bytes.
        Type << Global  PAGE 12  LINE 1016 >>

t_sgen_frd_sphere
    generic FRD spherical coordinates
        Type << Global  PAGE 4  LINE 313 >>

t_sgen_frolloff
    precipitation cell spectral roll-off frequency.  Max size: 400 bytes.
        Type << Global  PAGE 12  LINE 1011 >>

t_sgen_gainu
    The principle axes array factor pattern, direction u
        Type << Global  PAGE 16  LINE 1396 >>

t_sgen_gainv
    The principle axes array factor pattern, direction v
        Type << Global  PAGE 16  LINE 1401 >>

t_sgen_gc_radius
    target great circle radius
        Type << Global  PAGE 14  LINE 1158 >>

t_sgen_graz
    vector of grazing angles. Max size: 200,000 bytes.
        Type << Global  PAGE 11  LINE 976 >>

t_sgen_if_filter_response
    IF filter response matrix.  Fixed per run. Size: 180,224 bytes.
        Type << Global  PAGE 5  LINE 426 >>

t_sgen_ion_atten
    total one-way ionospheric attenuation in dB
        Type << Global  PAGE 14  LINE 1239 >>

t_sgen_ionosphere_model_coeffs
    Ionospheric model coefficients.  Max size: 1613 bytes.
           Type            Member
        t_sgen_faraday_rot   faraday_rot
        t_sgen_ion_atten     ion_atten
        t_sgen_seq_length    seq_length;
        t_sgen_dt_ion        dt_ion;
        t_spd_complex8       sparse_ionos[c_sgen_sparse_ionos];

        Type << Global  PAGE 15  LINE 1262 >>

t_sgen_iq_parameters
    I/Q parameters for antennas.  Fixed per run. Size: 176 bytes.
        Type << Global  PAGE 5  LINE 442 >>

t_sgen_iratio
    ratio of noise/jammer oversampling factor to clutter oversampling factor.  Fixed per run.
        Type << Global  PAGE 6  LINE 477 >>

t_sgen_isoelements
    vector containing the number of patches for each isodop. Max. size: 10,000 bytes
        Type << Global  PAGE 9  LINE 736 >>

t_sgen_iwindow
    interpolation window vector.  Fixed per run. Size: 8192 bytes.
        Type << Global  PAGE 6  LINE 483 >>

t_sgen_jammer_attenuation
    jammer atmospheric attenuation at start of burst
        Type << Global  PAGE 14  LINE 1200 >>

t_sgen_jammer_database
    jammer database. Max. size: 1200 bytes.
           Type                          Member
        t_sgen_jammer_validity         jammer_validity
        t_sgen_jammer_range            jammer_range
        t_sgen_jammer_attenuation      jammer attenuation

        Type << Global  PAGE 14  LINE 1216 >>

t_sgen_jammer_position
    Jammer position in frd coordinates typedef struct { t_sgen_jammer_position jam_pos_beg; t_
    sgen_jammer_position jam_pos_middle; t_sgen_jammer_position jam_pos_end; } t_sgen_jam_frd[c_
    cend_max_jammers]; /** position of jammers at beginning, middle, end of the time sequence.
```

```
==========================================================================================

t_sgen_jammer_position  (continued)
    Max size: 3600 bytes.
         Type << Global  PAGE 18  LINE 1548 >>

t_sgen_jammer_range
    jammer range at start of burst
         Type << Global  PAGE 14  LINE 1195 >>

t_sgen_jammer_samples
    number of jammer samples for the current CPI at oversampled rate
         Type << Global  PAGE 10  LINE 846 >>

t_sgen_jammer_validity
    records whether or not an individual jammer is within the FoV for the current CPI
         Type << Global  PAGE 14  LINE 1205 >>

t_sgen_jammer_window
    jammer window vector (windowj).  Size: 8192 bytes.
         Type << Global  PAGE 18  LINE 1563 >>

t_sgen_jstart_time
    Jammer active time segment start (tstart)
         Type << Global  PAGE 18  LINE 1583 >>

t_sgen_jstop_time
    Jammer active time segment stop (tstop)
         Type << Global  PAGE 18  LINE 1588 >>

t_sgen_k1
    constant angle (1/orbit radius).  This variable has no abstract meaning, but is used
    frequently throughout the algorithm
         Type << Global  PAGE 4  LINE 353 >>

t_sgen_k2
    constant (earth's rotational velocity * cos(SBR heading) * cos(SSP latitude)
         Type << Global  PAGE 9  LINE 757 >>

t_sgen_k3
    constant angle (earth's rotational velocity * cos(inclination) - SBR rotational velocity).
    This variable has no abstract meaning, but is used frequently throughout the algorithm
         Type << Global  PAGE 5  LINE 359 >>

t_sgen_max_dop_angle
    angle of maximum doppler in radians
         Type << Global  PAGE 9  LINE 763 >>

t_sgen_nchaffs
    number of chaff cells in chaff clutter database
         Type << Global  PAGE 13  LINE 1074 >>

t_sgen_nci_database
    Working variables for the current NCI, recalculated at the beginning of each NCI
         Type << Global  PAGE 9  LINE 714 >>

t_sgen_nci_srsd
    This structure defines information stored in the SRSD output file header block for each NCI
    in the run.  Max size: 264 bytes.
              TYPE                        MEMBER
          t_spd_card32                number_of_cpis
          t_sgen_adjusted_prf         prf;
          t_sgen_num_samples          number_of_samples(c_spd_max_CPI];

         Type << Global  PAGE 19  LINE 1651 >>

t_sgen_noise_coeffs
    Noise coefficients
                Type              Member
            t_sgen_noise_covar  noise_covar;
            t_sgen_uchol        uchol;
            t_sgen_prev_rf      prev_rf;

         Type << Global  PAGE 16  LINE 1372 >>

t_sgen_noise_covar
    noise covariance matrix.  Max size: 3872 bytes.
         Type << Global  PAGE 15  LINE 1282 >>

t_sgen_noise_power
    noise power at each antenna.  Size: 88 bytes.
         Type << Global  PAGE 16  LINE 1362 >>

t_sgen_noise_window
    noise window parameters.  Fixed per run. Size: 8192 bytes.
         Type << Global  PAGE 5  LINE 409 >>
```

```
==========================================================================================

t_sgen_nonlinearity_params
    table of nonlinearity parameters for the antennas.  Fixed per run. Size: 352 bytes.
        Type << Global  PAGE 6  LINE 488 >>

t_sgen_npplat
    vector representing number of clutter patches per latitude line. Fixed per run.  Size: 10,
    801 bytes.
        Type << Global  PAGE 6  LINE 505 >>

t_sgen_nprecip
    number of precipitation cells in precipitation clutter database
        Type << Global  PAGE 11  LINE 898 >>

t_sgen_nps
    number of elements in precipitation cells.  Max size: 400 bytes.
        Type << Global  PAGE 12  LINE 1021 >>

t_sgen_npstot
    total number of elements in precipitatione cells (summation of t_sgen_nps)
        Type << Global  PAGE 12  LINE 1032 >>

t_sgen_nsegin
    input segment length for the interpolation routine
        Type << Global  PAGE 18  LINE 1526 >>

t_sgen_num_ant
    number of antennas in current run.  Fixed per run.
        Type << Global  PAGE 5  LINE 394 >>

t_sgen_num_chan
    number of channels.  Fixed per run.
        Type << Global  PAGE 5  LINE 404 >>

t_sgen_num_iso_elements
    vector indicating the number of patches in each isodop. Max size: 10,000 bytes.
        Type << Global  PAGE 11  LINE 953 >>

t_sgen_num_isodops
    number of isodops in range-doppler map
        Type << Global  PAGE 9  LINE 768 >>

t_sgen_num_rs
    number of range-swaths.  Fixed per run.
        Type << Global  PAGE 6  LINE 457 >>

t_sgen_num_samples
    number of samples in the CPI
        Type << Global  PAGE 10  LINE 855 >>

t_sgen_number_of_cpis
    Total number of CPIs in the current run.  Fixed per run.
        Type << Global  PAGE 6  LINE 494 >>

t_sgen_omega_if
    A/D IF angular frequency.  Fixed per run.
        Type << Global  PAGE 5  LINE 437 >>

t_sgen_orbit_radius
    This variable specifies the SBR orbit radius in kilometers calculated from earth_radius +
    altitude.  Fixed per run.
        Type << Global  PAGE 4  LINE 342 >>

t_sgen_orbital_model_coeffs
    Orbital model coefficients.  Fixed per run.
            Members:
            time_zero        t_sgen_elapse_time (time at which ascending node
                                  crosses equatorial plane)
            omega_zero       t_spd_double (longitude of ascending node at time zero)

        Type << Global  PAGE 5  LINE 376 >>

t_sgen_out_off
    Offset of first sample to use for decimation.  Fixed per run.
        Type << Global  PAGE 5  LINE 420 >>

t_sgen_outoff
    first element to use from the interpolated segment
        Type << Global  PAGE 18  LINE 1521 >>

t_sgen_oversamples
    oversampled temporary matrix.  Max size: 540,672,000 bytes (roughly 516MB)
        Type << Global  PAGE 18  LINE 1532 >>

t_sgen_pc_aperture
    x and y dimensions for phase centers in each group.  Max size: 16 bytes.
        Type << Global  PAGE 15  LINE 1302 >>
```

```
t_sgen_pc_loc
   x and y locations for each phase centre.  Max size: 48 bytes.
       Type << Global  PAGE 15  LINE 1311 >>

t_sgen_percent_complete
   Run percentage complete at time of last progress message transmission
       Type << Global  PAGE 7  LINE 614 >>

t_sgen_precip_db
   precipitation database for the current CPI, generated as part of volumetric clutter database
           Type                 Member
        t_sgen_nprecip          nprecip
        t_sgen_frolloff         frolloff
        t_sgen_fpeak            fpeak
        t_sgen_nps              nps
        t_sgen_atmos_atten      atmos_atten
        t_sgen_npstot           npstot
        t_sgen_precip_rcs       precip_rcs
        t_sgen_pscatloc         pscatloc

       Type << Global  PAGE 12  LINE 1057 >>

t_sgen_precip_dbase
   precipitation database generated per cpi. Max. size: 2400 bytes.
           type             member
        t_spd_real       cell_azimuth_extent_x;
        t_spd_real       cell_azimuth_extent_y;
        t_spd_real       cell_elevation_x;
        t_spd_real       cell_elevation_y;
        t_spd_real       cell_attenuation;
        t_spd_real       cell_thickness;

       Type << Global  PAGE 11  LINE 911 >>

t_sgen_precip_rcs
   radar cross section of precipitation cells.  Max size: 40,000 bytes.
       Type << Global  PAGE 12  LINE 1037 >>

t_sgen_prev_rf
   RF of previous CPI (RF[-1]=0)
       Type << Global  PAGE 15  LINE 1293 >>

t_sgen_pscatloc
   precipitation scatterer locations. Max size: 120,000 bytes.
       Type << Global  PAGE 12  LINE 1042 >>

t_sgen_pulse_compression_replica
   pulse compression replica for the current dwell.  This is before decimation for storage.
   Max size: 80,000 bytes.
       Type << Global  PAGE 10  LINE 823 >>

t_sgen_radar_wavelength
   radar wavelength for the current CPI
       Type << Global  PAGE 9  LINE 742 >>

t_sgen_range
   vector of mid-swath ranges.  Max size: 200,000 bytes.
       Type << Global  PAGE 12  LINE 981 >>

t_sgen_range_bins
   number of range bins in the current MCI,
       Type << Global  PAGE 8  LINE 694 >>

t_sgen_range_quant
   range quantization.  Fixed per run.
       Type << Global  PAGE 6  LINE 452 >>

t_sgen_receive_end_time_ref
   receive end time of current CPI
       Type << Global  PAGE 10  LINE 841 >>

t_sgen_receive_ref_time
   receive reference time for the MCI (t_rx_ref)
       Type << Global  PAGE 8  LINE 704 >>

t_sgen_receive_start_time_ref
   receive start time of current CPI
       Type << Global  PAGE 10  LINE 837 >>

t_sgen_replica_interpolation_factor
   replica interpolation factor.  Fixed per run.
       Type << Global  PAGE 5  LINE 367 >>

t_sgen_rot_matrix
   general type used for an angular rotation matrix
       Type << Global  PAGE 4  LINE 318 >>
```

```
==========================================================================================

t_sgen_rotational_velocity
    SBR rotational velocity in radians per second.  Fixed per run.
         Type << Global  PAGE 4  LINE 348 >>

t_sgen_run_constants
    computed constants used for the whole run
               Members:                       Types:
                 number_of_cpis                t_sgen_number_of_cpis
                 orbit_radius                  t_sgen_orbit radius
               rotational_velocity           t_sgen_rotational_velocity
                 k1                            t_sgen_k1
                 k3                            t_sgen_k3
                 velocity                      t_sgen_velocity
               slant_range                   t_sgen_slant_range
               orbital_model_coeffs          t_sgen_orbital_model_coeffs
               number_of_antennas            t_sgen_num_ant
               number_of_channels            t_sgen_num_chan
               noise_window                  t_sgen_noise_window
               first_channel                 t_sgen_first_channel
               out_off                       t_sgen_out_off
               if_filter_response            t_sgen_if_filter_response
               differentiator                t_sgen_differentiator
               omega_if                      t_sgen_omega_if
               iq_parameters                 t_sgen_iq_parameters
               ad_parameters                 t_sgen_ad_parameters
               range_quant                   t_sgen_range_quant
               num_rs                        t_sgen_num_rs
                 elevation_angles              t_sgen_elevation_angles
                 replica_interpolation_factor  t_sgen_replica_interpolation_factor
                 trop_ref_db                   t_sgen_trop_ref_db
                 antenna_idx                   t_sgen_antenna_idx
                 iratio                        t_sgen_iradio
                 iwindow                       t_sgen_iwindow
                 nonlinearity_params           t_sgen_nonlinearity_params
                 npplat                        t_sgen_npplat
                 clutter_map_db                t_sgen_clutter_map_db

        Type << Global  PAGE 7  LINE 577 >>

t_sgen_run_parameters
    parameters which maintain their overall integrity across the course of the run
               Type                           Member
            t_sgen_percent_complete          percent_complete
            t_sgen_current_dwell             current_dwell
            t_sgen_current_nci               current_nci
            t_sgen_current_cpi               current_cpi
            t_sgen_cpi_counter               cpi_counter
            t_sgen_all_seeds                 all_seeds

        Type << Global  PAGE 8  LINE 640 >>

t_sgen_run_state
    signal generation internal running state.  This is separated out of RSS_State to allow the
    tasks to maintain intermediate states "aborting" and "terminating"
        Type << Global  PAGE 3  LINE 246 >>

t_sgen_sbr_heading
    instantaneous SBR heading
        Type << Global  PAGE 9  LINE 747 >>

t_sgen_sbr_orbit_position
    sparce sequence of SBR positions during the current CPI. Max size: 36,000 bytes.
            TYPE                   MEMBER
        t_sgen_ecr_vector       ecr_position; position in ECR caresians
        t_spd_real             sin_heading;  sin of the instantaneous heading
        t_spd_real             cos_heading;  cos of the instantaneous heading
        t_spd_real             sin_lat;      sin of the SSP latitude
        t_spd_real             cos_lat;      cos of the SSP latitude
        t_spd_real             sin_lon;      sin of the SSP longitude
        t_spd_real             cos_lon;      cos of the SSP longitude

        Type << Global  PAGE 17  LINE 1453 >>

t_sgen_seed
    seed parameter used for generating a random number
        Type << Global  PAGE 4  LINE 329 >>

t_sgen_seg
    noise generation subsegment vector.  Size 8192 bytes.
        Type << Global  PAGE 19  LINE 1611 >>

t_sgen_seg_int
    Jammer segment interval (seg_int)
        Type << Global  PAGE 18  LINE 1573 >>
```

```
t_sgen_seg_time
    Jammer segment time (seg_time);
        Type << Global  PAGE 18  LINE 1578 >>

t_sgen_seq_length
    the length of the sparse ionospheric sequence for the current burst, in samples
        Type << Global  PAGE 14  LINE 1244 >>

t_sgen_slant_range
    slant range to horizon. Fixed per run.
        Type << Global  PAGE 6  LINE 472 >>

t_sgen_spec
    noise generation frequency-domain vector.  Size 8192 bytes.
        Type << Global  PAGE 19  LINE 1606 >>

t_sgen_sphere_pos_vect
    Column vector of position in spherical coordinates
        Type << Global  PAGE 4  LINE 293 >>

t_sgen_srsd_header
    this structure defines the header for the SRSD output file, providing information needed to
    playback the generated data
        TYPE                    MEMBER
        char                    satdef_pathname[c_spd_path_size];
        char                    envdef_pathname[c_spd_path_size];
        t_sgen_num_chan         number_of_channels;
        t_sgen_number_of_cpis   number_of_cpis;
        t_sgen_cpi_srsd         cpi_info[c_sgen_max_cpis];

        Type << Global  PAGE 19  LINE 1670 >>

t_sgen_srsd_int
    The base type for stored radar samples
        Type << Global  PAGE 20  LINE 1701 >>

t_sgen_srsd_sample
    Definition for one single complex radar sample, as output to the SRSD file and the A/D
    intermediate product file
        Type << Global  PAGE 20  LINE 1711 >>

t_sgen_srsd_samples
    simulated radar signal data for one CPI (final, no oversampling, but still floating point).
    Max size: 135,168,000 bytes (roughly 129MB).
        Type << Global  PAGE 19  LINE 1640 >>

t_sgen_sweep_mod
    Jammer sweep modifier
        Type << Global  PAGE 18  LINE 1568 >>

t_sgen_t_rot_angle
    target rotation angle at current time
        Type << Global  PAGE 14  LINE 1168 >>

t_sgen_target_attenuation
    target atmospheric attenuation at start of burst
        Type << Global  PAGE 13  LINE 1147 >>

t_sgen_target_database
    target database.  Max. size: 1200 bytes.
            Type                        Member
        t_sgen_target_validity        target_validity
        t_sgen_target_range           target_range
        t_sgen_target_attenuation     target_attenuation

        Type << Global  PAGE 14  LINE 1178 >>

t_sgen_target_range
    target range at start of burst
        Type << Global  PAGE 13  LINE 1142 >>

t_sgen_target_validity
    records whether or not an individual target is within the FoV for the current CPI
        Type << Global  PAGE 13  LINE 1152 >>

t_sgen_trans_end_time
    transmit end time of current CPI
        Type << Global  PAGE 10  LINE 833 >>

t_sgen_trans_start_time
    transmit start time of current CPI
        Type << Global  PAGE 10  LINE 829 >>

t_sgen_trop_ref_db
    tropospheric refraction database.  Rows -> altitude; columns -> angles each entry represents
    the refraction induced range error.  Fixed per run. Size: 2400 bytes.
```

```
==============================================================================================================
t_sgen_trop_ref_db  (continued)
      Type << Global  PAGE 5  LINE 387 >>

 t_sgen_tx_CFA_antenna_pattern
    transmit antenna pattern.  This matrix represents four generated look-up tables (GainU,
    GainV, EgainU, EgainV) of differing length, and will be revised.  Here for sizing only.  Max
    size: 65,536 bytes.
         Type << Global  PAGE 16  LINE 1389 >>

 t_sgen_uchol
    Cholesky factor matrix associated with the noise covariance matrix. Max size: 3872 bytes.
         Type << Global  PAGE 15  LINE 1287 >>

 t_sgen_velocity
    SBR velocity.  Fixed per run.
         Type << Global  PAGE 6  LINE 467 >>

 t_sgen_vol_clutter_db
    Volumetric clutter database (precipitation database + chaff database)
         Type << Global  PAGE 13  LINE 1131 >>

 t_sgen_waveform_coeffs
    Waveform coefficients
              Type                               Member
         t_sgen_pulse_compression_replica      pulse_compression_replica;
         t_sgen_trans_start_time               trans_start_time;
         t_sgen_trans_end_time                 trans_end_time;
         t_sgen_receive_start_time_ref         receive_start_time_ref;
         t_sgen_receive_end_time_ref           receive_end_time_ref;
         t_sgen_jammer_samples                 jammer_samples;
         t_sgen_clutter_samples                clutter_samples;
         t_sgen_num_samples                    num_samples;
         t_sgen_adjusted_prf                   adjusted_prf;

       Type << Global  PAGE 10  LINE 875 >>

t_spd_boolean
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_card16
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_card32
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_card8
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_complex8
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_double
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_int16
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_int32
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_int8
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_lat
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_location
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_lon
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>

t_spd_real
    ** No Dictionary Definition Given **
       Type << Global  File Not Printed >>
```

E-36

```
==============================================================================================

t_spd_sat_dir
    ** No Dictionary Definition Given **
        Type << Global  File Not Printed >>

t_spd_status
    ** No Dictionary Definition Given **
        Type << Global  File Not Printed >>
```

```
                              Requirements Traceability
=====================================================================================================
SUBJECT
    PAGE NO.   LINE NO.      ENCLOSING FUNCTION

   NO MARKS TO REFERENCE
```

```
                                        TBD CROSS REFERENCE REPORT
================================================================================================================
=================================
PAGE NO.   LINE NO.          ENCLOSING FUNCTION

  NO TBDs TO REFERENCE
```

```
                              ERROR SUMMARY
=================================================================================
PAGE NO.   LINE NO.    ERROR TEXT

  NO ERRORS TO REPORT
```

1.6 RPCGEN Data header file for Playback (SPMT.X)

Data header files relating to RPCs must be compatible with the RPCGEN utility, and therefore require a format which is not easily compatible with CDADL. For the sake of clarity, this document contains a listing of spmt.x, the RPCGEN source file, rather than a CDADL listing of spmt.h.

```
/*====================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:                                   spmt_intf.x - Playback Manager <-> DSP Interface header file

Identification: [@(#)]

SCCS Id:        @(#)spmt_intf.x          1.1 93/04/27

Responsible Organization:

                Raytheon Canada Limited

Purpose:  This package declares RPC interfaces (constants and message
                                             types) between the Playback Manager task and the DSP Interface
                task.

                                             This file is written in RPC data description language in XDR
                                             (eXternal Data Representation) standard. This file will use
                                             "rpcgen" utility to generate the following two files:

                                             spmt_intf.h    - global include file for both tasks
                                             spmt_xdr.c     - XDR source file which contains required XDR
                                                      conversion routines to support the RPC
                                                      message data types.

                                             The generated files are needed for building the executables for
                                             both the Playback Manager Task and the DSP Interface task.

Designer:       Marg Baskin

Change History:

    SCR   Date         Author   Change made
    ---   -----------  ------   ----------------------------------------
    n/a   08-APR-1993   mdb     Initial Release

=====================================================================*/

%/*---------------------------------------------------------------------- **
%**                                                                        **
%**    DEFINE CONSTANTS                                                    **
%**                                                                        **
%**-----------------------------------------------------------------------*/
%/*
% * The RPC program and version definitions
% *     *** TBD in concert with MDA for consistency  ***
% */

#ifdef RPC_HDR
%#ifndef spmt_intf_h
%#define spmt_intf_h

%/*-------------------------------------------------------------------
% *
% *   Data structure for SRSD commands.
% *
% *   source:                               Playback Manager task
% *   sink:                                 DSP Interface task
% *
% *   The SRSD commands consist of a command type and command
% *   parameters.  The command type tells which command parameters are
% *
% *   The SRSD command type can be either:
% *
% *       1) start_playback_command
% *       2) abort_playback_command
% *       3) load_playback_command -- with parameters
% */

enum t_spmt_srsdcmd_type
{
   c_spmt_start_playback,                   /* start playback */
   c_spmt_abort_playback,                   /* abort playback */
   c_spmt_load_playback                        /* load SRSD file */
};


%/*-------------------------------------------------------------------
% *
% *   Data structure for SRSD Command Parameters
% *
% *   These parameters are passed with the load playback command to indicate
% *   which simulated radar signal data (SRSD) is to be played back, and how many
% *   iterations are used.
% *
% *      SRSD_pathname  - pathname (full pathname from the root to the
% *                       particular file) which indicates where the SRSD
% *                       file is stored.  If the file is stored in a
% *                       disk on different machine, then the disk must be
% *                       remote-mounted via NFS.
% *                       RSS uses the provided SRSD_pathname file name
% *                       to playback the simulated signal data.
% *
% *      iteration_count       -  number of iterations used for a run
% *
% */

struct t_spmt_playback_parms
{
   char                                SRSD_pathname[c_spd_path_size]; /* pathname of SRSD file */
   t_spd_int32                         iteration_count;         /* number of interations */
};

union t_spmt_playback_commands switch (enum t_spmt_srsdcmd_type command_type)
```

```
{
   case c_spmt_start_playback:                     /* start playback command */
      void;

   case c_spmt_abort_playback:                  /* abort playback command */
      void;

   case c_spmt_load_playback:
      t_spmt_playback_parms playback_parms;  /* load playback parameters */

};

%/*================================================================
% *
% *   DSP Interface response message definitions (return status from DI).
% *
% *----------------------------------------------------------------
% *
% */
enum t_spmt_plbrsp_type
{
   c_spmt_playback_ready,                   /* response to load playback */
   c_spmt_playback_aborted,                 /* response to abort playback */
   c_spmt_playback_complete                 /* response to start playback; sent per iteration */
};

%/*----------------------------------------------------------------
% *
% *   Integers enumerating status return codes from DSP Interface
% */
enum t_spmt_simstat_type
{
   c_spmt_completed_normally,               /* activity completed normally */
   c_spmt_in_progress,                  /* activity in progress */
   c_spmt_terminated_with_error             /* activity terminated with error */
};

enum t_spmt_abort_status
{
   c_spmt_abort_completed,                  /* successfully aborted the current run  */
   c_spmt_abort_failed                          /* failed to abort playback */

};

%/*----------------------------------------------------------------
% *
% *   Data structure for return status from DI
% *
% *   source:   DSP Interface Task
% *   sink:                          Playback Manager Task
% *
% *   Playback status consists of:
% *
% *      status              -  indicates whether the activity was
% *                             completed normally, or was terminated with error.
% *
% *
% *      In the case where playback activity is a normally terminated playback iteration
% *      this message will also include the actual elapsed
% *      time of data playback for the current iteration.
% *
% *      iteration_no        -  iteration number of data segment just completed
% *                             data playback
% *      elapsed_time        -  actual elapsed time of data playback for
% *                             the current iteration
% *
% */
struct t_spmt_playback_status
{
   t_spmt_simstat_type status;                /* current RSS simulation playback status */
   t_spd_int32                        iteration_no;       /* current iteration in the playback activity */
   t_spd_real                         elapsed_time;       /* indicates the elapsed time of data
                                              playback for the current iteration
                                              (interation_no) */

};

%/*----------------------------------------------------------------
% *
% *
% *   Data structure for Playback Responses message.
% *
% *   source:   DSP Interface Task
% *   sink:                          Playback Manager Task
% *
% *   The Playback Response message consists of a response type and response
% *   parameters.  The response type tells which response parameters are
% *   sent.
% *
% *   The Playback Response response type can be:
% *
% *        1) playback_complete - playback status parameters follow
% *        2) playback_aborted - abort status parameters follow
% *        3) playback_ready - playback status parameters follow
% *
% *
% */
union t_spmt_playback_rsp switch (enum t_spmt_plbrsp_type rsp_type)
{
   case c_spmt_playback_complete:
      t_spmt_playback_status             playback_status;/* start playback response */

   case c_spmt_playback_aborted:
      t_spmt_abort_status                abort_status;  /* abort playback response */

   case c_spmt_playback_ready:
      t_spmt_simstat_type                load_status;/* load playback response */

};

#ifdef RPC_HDR
%#endif
#endif
```

```
typedef struct
{
   t_sgen_number_of_cpis                number_of_cpis;
    _sgen_orbit_radius                  orbit_radius;
   t_sgen_rotational_velocity           rotational_velocity;
   t_sgen_k1                            k1;
   t_sgen_k3                            k3;
   t_sgen_velocity                      velocity;
   t_sgen_range_quant                   range_quant;
   t_sgen_slant_range                   slant_range;
   t_sgen_orbital_model_coeffs          orbital_model_coeffs;
   t_sgen_num_ant                       number_of_antennas;
   t_sgen_num_chan                      number_of_channels;
   t_sgen_noise_window                  noise_window;
   t_sgen_jammer_window                 jammer_window;
   t_sgen_first_channel                 first_channel;
   t_sgen_antenna_index                 antenna_index;
   t_sgen_out_off                       out_off;
   t_sgen_if_filter_response            if_filter_response;
   t_sgen_differentiator                differentiator;
   t_sgen_omega_if                      omega_if;
   t_sgen_ad_parameters                 ad_parameters;
   t_sgen_num_rs                        num_rs;
   t_sgen_elevation_angles              elevation_angles;
   t_sgen_trop_ref_db                   trop_ref_db;
   t_sgen_iratio                        iratio;
   t_sgen_iwindow                       iwindow;
   t_sgen_npplat                        npplat;
   t_sgen_clutter_map_db                clutter_map_db;
    _sgen_rand                          RandU;
   t_sgen_rand                          RandV;
   t_sgen_3x3_matrix                    Atrans1;
   t_sgen_3x3_matrix                    Atrans2;
   t_sgen_3x3_matrix,                   transl_frd_to_rra;
   t_sgen_3x3_matrix                    transl_rra_elev;
   t_sgen_3x3_matrix                    transl_rra_to_frd;
   t_sgen_data_capt_chnl_matrix         capt_chnl_matrix;
   t_sgen_data_capt_chnl_count          capt_chnl_count;
} t_sgen_run_constants;

typedef struct
{
   t_sgen_current_dwell       current_dwell;
   t_sgen_current_nci         current_nci;
   t_sgen_current_cpi         current_cpi;
   t_sgen_cpi_counter         cpi_counter;
   t_sgen_trans_start_time    trans_start_time;
   t_sgen_trans_end_time      trans_end_time;
   t_sgen_all_seeds           all_seeds;
   t_spd_int32                ip_time_stop;
   t_spd_int32                ip_num_range_samps;
   t_spd_int32                ip_clutter_range_start;
   t_spd_int32                ip_clutter_range_stop;
   t_spd_int32                ip_clutter_range_step;
    _spd_int32                ip_jammer_range_start;
    _spd_int32                ip_jammer_range_stop;
   t_spd_int32                ip_jammer_range_step;
   t_spd_int32                ip_range_start;
   t_spd_int32                ip_range_stop;
   t_spd_int32                ip_range_step;
   t_spd_int32                  num_int_prod_sample_per_chanl;
} t_sgen_run_parameters;


typedef struct
{
   t_sgen_dwell_database            dwell_database;
   t_sgen_nci_database              nci_database;
   t_sgen_cpi_database              cpi_database;
   t_sgen_waveform_coeffs           waveform_coeffs;
   t_sgen_attenuation_database      attenuation_database;
   t_sgen_ec_database               earth_clutter_database;
   t_sgen_vol_clutter_db            vol_clutter_db;
   t_sgen_target_database           target_database;
   t_sgen_jammer_database           jammer_database;
   t_sgen_ionosphere_model_coeffs   ionosphere_model_coeffs;
   t_sgen_antenna_database          antenna_database;
   t_sgen_noise_coeffs              noise_coeffs;
   t_sgen_sbr_orbit_position        orbit_position;
} t_sgen_burst_database;

typedef struct
{
   t_sgen_beam_centre_latitude     beam_centre_latitude;
   t_sgen_beam_centre_longitude    beam_centre_longitude;
} t_sgen_dwell_database;


typedef struct
{
   t_sgen_range_bins          range_bins;
   t_sgen_coh_burst_time      coh_burst_time;
   t_sgen_receive_ref_time    receive_ref_time;
   t_sgen_ant_trans           ant_trans;
   t_sgen_trans_rra_to_aux    trans_rra_to_aux[c_sgen_max_antennas];
   t_sgen_cart_vector         antenna_location[c_sgen_max_antennas];
} t_sgen_nci_database;

typedef struct
{
   t_sgen_dop_quant              doppler_quant;
   t_sgen_equi_range_vector      equi_range_vector;
   t_sgen_isoelements            isoelements;
   t_sgen_radar_wavelength       radar_wavelength;
   t_sgen_sbr_heading            sbr_heading;
   t_sgen_k2                     k2;
   t_sgen_max_dop_angle          max_dop_angle;
   t_sgen_num_isodops            num_isodops;
   t_sgen_delta_rf               delta_rf;
   t_sgen_SSP_current            SSP_current;
   t_sgen_current_direction      current_direction;
   t_sgen_cart_vector            SBR_pos_in_ecr;
   t_sgen_3x3_matrix             rotation_3FN;
   t_sgen_3x3_matrix             rotation_23NE;
} t_sgen_cpi_database;


typedef struct
{
   t_sgen_pulse_compression_replica    pulse_compression_replica;
```

```
    t_sgen_receive_start_time_ref       receive_start_time_ref;
    t_sgen_receive_end_time_ref         receive_end_time_ref;
    t_sgen_first_sample_time            first_sample_time;
     _sgen_jammer_samples               jammer_samples;
    t_sgen_clutter_samples              clutter_samples;
    t_sgen_num_samples                  num_samples;
    t_sgen_adjusted_prf                 adjusted_prf;
    t_sgen_waveform                     waveform;
    t_sgen_waveform_length              waveform_length;
 } t_sgen_waveform_coeffs;

typedef struct
{
    t_sgen_nprecip           nprecip;
    t_sgen_precip_dbase      *precip_dbase;
    t_sgen_at_attenuation    at_attenuation;
} t_sgen_attenuation_database;

typedef struct
{
    t_sgen_cpc           cpc;
    t_sgen_cpa           cpa;
    t_sgen_cpa_row_index cpa_row_index;
    t_sgen_graz         graz;
    t_sgen_range         range;

} t_sgen_ec_database;


  def struct
{
    t_sgen_precip_db    precip_db;
    t_sgen_chaff_db     chaff_db;
} t_sgen_vol_clutter_db;


typedef struct
{
    t_sgen_target_validity      target_validity;
    t_sgen_target_range         target_range;
    t_sgen_target_attenuation   target_attenuation;
} t_sgen_target_database;


typedef struct
{
    t_sgen_jammer_validity      jammer_validity;
    t_sgen_jammer_range         jammer_range;
    t_sgen_jammer_attenuation   jammer_attenuation;
} t_sgen_jammer_database;

typedef struct
{
    t_sgen_faraday_rot     faraday_rot;
     _sgen_ion_atten       ion_atten;
    t_sgen_seq_length      seq_length;
    t_sgen_dt_ion          dt_ion;
    t_spd_complex8         sparse_ionos[c_sgen_sparse_ionos];
} t_sgen_ionosphere_model_coeffs;
```

```
typedef struct
{
   union
     {
        t_sgen_CFA_antenna_database CFA;
        t_sgen_RRA_antenna_database RRA;
     } main_antenna;

   t_sgen_CFA_antenna_gains aux_antenna[c_spd_max_aux];
} t_sgen_antenna_database;


typedef struct
{
   t_sgen_noise_covar  noise_covar;
   t_sgen_uchol        uchol;
   t_sgen_prev_rf      prev_rf;
} t_sgen_noise_coeffs;


typedef struct
{
   t_sgen_cart_vector   ecr_cart_vector;
   t_spd_real           sin_heading;
   t_spd_real           cos_heading;
   t_spd_real           sin_lat;
   t_spd_real           cos_lat;
   t_spd_real           sin_lon;
   t_spd_real           cos_lon;
} t_sgen_sbr_orbit_position;
```

E2 Signal Generation Manager (SSGM) CSU

THIS PAGE INTENTIONALLY LEFT BLANK

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□                      □□□□
□□□□                      □□□□
□□□□                      □□□□
  □□                        □□
  □□     CDADL PROCESSOR          □□
  □□                        □□
  □□                        □□
  □□      VERSION 2.4.6        □□
  □□                        □□
  □□   SERIAL NUMBER: C38U0011      □□
  □□                        □□
  □□     AUTHORIZED USER:         □□
  □□      Raytheon - Canada      □□
  □□                        □□
  □□                        □□
  □□                        □□
  □□                        □□
  □□                        □□
□□□□                      □□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

1 FILE ssgm.c
3 FILE ssgm_cleanup.c
4 FILE ssgm_tc_inbound.c
6 FILE ssgm_initialize.c
7 REPORT Requirements Traceability
8 REPORT INVOCATION TREE
9 REPORT ERROR SUMMARY

```
/*========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      ssgm.c

Identification: [@(#)]

SCCS Id:        @(#)ssgm        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author  Change made
---   ----------- ------  --------------------------------------
n/a   08-APR-1993  mdb     Initial Release

-----------------------------------------------------------------------

CSU_NAME:       ssgm (Signal Generation Manager)

FUNCTION:       ssgm_main

PURPOSE:        This task starts up the Signal Generation package, then
functions as an RPC listener for the package.  When
the RSS is started by the Testbed Controller, it spawns the
main Signal Generation task (sgen), then listens forever for
RPCs from the testbed controller, and passes them on to sgen
via a message queue (in order to let sgen avoid the hassles of
dealing with RPCs.)

ssgm also listens for RPCs from sgen and forwards them to
the testbed controller.


INPUT                                   SOURCE
-----                                   ------

Start generation command                   Testbed Controller
Abort generation command                    Testbed Controller
Shutdown generation command                   Testbed Controller
Status message                            Testbed Controller
Ping message                              Testbed Controller
Start generation response                  sgen
Abort generation response                   sgen

OUTPUT                                    DESTINATION
------                                  -----------

RPC acknowledge                            Testbed Controller
```

```
StartSignalGeneration message                    sgen
AbortSignalGeneration message                     sgen

=========================================================================*/


#include "ssgm_priv.h"

void ssgm_main ()
    /*% Signal Generation Manager Task */

{

/*|

    Use f_ssgm_initialize() to initialize the signal generation package

    IF return status != s_spd_success THEN
    |   use exit() to terminate the task; startup has failed
    ENDIF

    ### Use fork() to split the task into two threads ###

    ### TC -> sgen thread ###

    IF (fork() = 0) THEN
    |
    |   Use svc_run() to wait for incoming messages from the TC
    |   ;Message handler is f_ssgm_tc_inbound()
    |
    |
       |  ### sgen -> TC thread ###
       |
       ELSE
       |
       |  FOR (;;)
       |  LOOP
       |  |
       |  |  use a blocking msgrcv(t_ssgm_msgqid...) to wait for a message from sgen
       |  |  use clnt_call() to forward message to the TC
       |  |
       |  ENDLOOP
       |
       ENDIF

*/

       /* Local macros */

       /* Local constants */

       /* Local Structure Definitions */

       /* Local variables */

       /* Code */

}

/* end of ssgm_main.c */
```

```
/*=====================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      ssgm_cleanup.c

Identification: [@(#)]

SCCS Id:        @(#)ssgm_cleanup        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date         Author   Change made
---  ----------  ------  ------------------------------------
n/a  08-APR-1993   mdb      Initial Release

------------------------------------------------------------------

CSU_NAME:       ssgm (Signal Generation Manager)

FUNCTION:       i_ssgm_cleanup

PURPOSE:        This function performs clean-up activities prior to the
termination of ssgm's child task.
This includes deallocation of shared memory and release
of the message queue.

It is set up as an interrupt handler, and is entered
upon receipt of a "kill" signal from the parent task.


=====================================================================*/

i_ssgm_cleanup ()
    /*% i_ssgm_cleanup function */

{

/*|

    use shmdt() to detach shared memory
    delete the IPC message queue
    use exit() the process

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=============================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      ssgm_tc_inbound.c

Identification: [@(#)]

SCCS Id:        @(#)ssgm_tc_inbound     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------  ------  -----------------------------------
n/a  08-APR-1993   mdb     Initial Release

-------------------------------------------------------------------------

CSU_NAME:       ssgm (Signal Generation Manager)

FUNCTION:       f_ssgm_tc_inbound ()

PURPOSE:        This function processes an RPC received from the TC and
forwards it to sgen.


=============================================================================*/

void f_ssgm_tc_inbound ()
    /*% f_ssgm_tc_inbound process */
{

/*|

    Use svc_getargs() to read the incoming message from the TC

    SWITCH on t_crgi_genstat_type
    |
    |  CASE c_cget_start_gencmd
    |  CASE c_cget_abort_gencmd
    |  |
    |  |  Use svc_sendreply() to send xdr_void to the TC
    |  |  Use msgsnd() to forward the message to sgen
    <-------break
    |  |
    |  CASE c_cget_status_cmd
    |  |
    |  |  Use svc_sendreply() to send t_ssgm_rss_state
    <-------break
```

```
   | |
   | CASE c_cget_ping_cmd
   | | Use svc_sendreply() to send xdr_void to the TC
   <-------break
   | |
   | CASE c_cget_shutdwn_gencmd
   | |
   | | Use svc_sendreply() to send xdr_void to the TC
   | | Use msgsnd() to forward the message to sgen
   | | use kill() to terminate the other process thread
   | | Use exit() to terminate the process
   | |
   ENDSWITCH

*/

   /* Local macros */

   /* Local constants */

   /* Local Structure Definitions */

   /* Local variables */

   /* Code */


}

/* end of ssgm_tc_inbound.c */
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      ssgm_initialize.c

Identification: [@(#)]

SCCS Id:        @(#)ssgm_initialize    1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:      Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------- ------  ---------------------------------------
n/a  08-APR-1993   mdb     Initial Release

-----------------------------------------------------------------------

CSU_NAME:       ssgm (Signal Generation Manager)

FUNCTION:       f_ssgm_initialize ()

PURPOSE:        This process performs initialization functions for ssgm.

=========================================================================*/

void f_ssgm_initialize()
    /* f_ssgm_initialize function */
{

/*I

    Get node names for the TC (CNTL_NODE) and RSS (RSS_NODE) from
    environment variables

    Use rsh() to spawn sgen task

    use msgget() to create an intertask queue to share with sgen
    IF return status indicates a failure THEN
<-------return (s_spd_failure)
    ENDIF

    ### initialize RPC interface to testbed controller ###

    Use pmap_unset() to clean up previous incarnations of the RPC map
    Use svc_register() to register the TCP transport services for the TC and sgen
    Use svctcp_create() to create TCP receiving ports for the TC and sgen
    Use clnt_create() to create a client object for the TC
    IF no errors reported THEN
<-------return (s_spd_success)
    ELSE
<-------return (s_spd_failure)
    ENDIF
    attach shared memory for passing t_sgen_RSS_state

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local variables */

    /* Code */


}

/* end of ssgm_initialize.c */
```

SUBJECT
PAGE NO. LINE NO. ENCLOSING FUNCTION

NO MARKS TO REFERENCE

1 60

INVOCATION TREE

```
=================================================================================
================
PAGE NO.   LINE NO.


    1        67      ssgm_main
    6       325      |  f_ssgm_initialize
    4       232      |  f_ssgm_tc_inbound

    3       169      i_ssgm_cleanup
```

ERROR SUMMARY

==
==================

PAGE NO. LINE NO. ERROR TEXT

NO ERRORS TO REPORT

2 22

```
                                                     TBD CROSS REFERENCE REPORT
===============================================================================================================================
=================================
PAGE NO.   LINE NO.          ENCLOSING FUNCTION

  NO TBDs TO REFERENCE
```

THIS PAGE INTENTIONALLY LEFT BLANK

E3 Signal Generation Controller (SGEN) CSU

THIS PAGE INTENTIONALLY LEFT BLANK

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□                     □□□□
□□□□                     □□□□
□□□□                     □□□□
 □□                       □□
 □□         CDADL PROCESSOR           □□
 □□                       □□
 □□                       □□
 □□          VERSION 2.4.6        □□
 □□                       □□
 □□       SERIAL NUMBER: C38U0011         □□
 □□                       □□
 □□         AUTHORIZED USER:          □□
 □□         Raytheon - Canada         □□
 □□                       □□
 □□                       □□
 □□                       □□
 □□                       □□
 □□                       □□
□□□□                     □□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

1 FILE sgen.c
4 FILE sgen_abort.c
5 FILE sgen_initialize.c
6 FILE sgen_failed.c
7 FILE sgen_terminate.c
8 REPORT Requirements Traceability
9 REPORT INVOCATION TREE
10 REPORT ERROR SUMMARY

```
/*===========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:     sgen.c

Identification: [@(#)]

SCCS Id:       @(#)sgen       1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:      Marg Baskin

Change History:

SCR   Date        Author  Change made
---  ----------  -----  --------------------------------
n/a  08-APR-1993  mdb    Initial Release

---------------------------------------------------------------------

CSU_NAME:      sgen (Signal Generation Controller)

FUNCTION:      sgen_main

PURPOSE:       This is the main control segment of the signal generation
task.  It handles external communications (to ssgm), task
initialization, shutdown and clean-up.

This task is started by ssgm as part of its initialization
functions, and remains active/idle until it receives an
explicit shutdown command from the terminal controller.

INPUT                                        SOURCE
-----                                        ------
StartSignalGeneration message                 ssgm
AbortSignalGeneration message                 ssgm
ShutdownSignalGeneration message               ssgm

OUTPUT                                       DESTINATION
------                                       -----------
SignalGenerationResponse                      ssgm
AbortGenerationResponse                       ssgm
===========================================================================*/

#include "sgen_priv.h"

void sgen_main()
    /*% Signal Generation Controller Main Task */

{
```

```

/*|

    use f_sgen_initialize() to setup

    DO
    |
    |  IF t_sgen_run_state = s_sgen_idle THEN
    |  |  use a blocking msgrcv() to receive a message from ssgm
    |  ELSE
    |  |  use a non-blocking msgrcv() to receive a message from ssgm,
    |  |  when one is present
    |  ENDIF
    |
    |  IF a message is present THEN
    |  |
    |  |  SWITCH on command type
    |  |  |
    |  |  |  CASE c_crgi_start_gencmd
    |  |  |  |
    |  |  |  |  IF t_crgi_RSS_state = s_sgen_idle THEN
    |  |  |  |  |  t_crgi_RSS_state = s_sgen_running
    |  |  |  |  |  t_sgen_run_state = s_sgen_running
    |  |  |  |  |  use f_sutl_olog(t_crgi_sgen_log...) to initialize the error log
    |  |  |  |  |  use f_sutl_elog() to log "starting generation run"
    |  |  |  |  |  cast satdef_pathname, envdef_pathname, output_dir, gen_ident
    |  |  |  |  |  to shared global data
    |  |  |  |  |  set t_sgen_active_status = s_sgen_srun_working
    |  |  |  |  |  t_crgi_genrsp.rsp_type = c_crgi_start_genrsp
    |  |  |  |  |  t_crgi_gen_status.status = c_crgi_in_progress
    |  |  |  |  |  t_crgi_gen_status.percent = 0
| | | | | |  use msgsnd() to send the message to ssgm
| | | | | |  use fork() to start srun
| | | | | |
| | | | |  ELSE
| | | | | |  t_crgi_genrsp.rsp_type = c_crgi_start_genrsp
| | | | | |  t_crgi_gen_status.status = c_crgi_in_progress
| | | | | |  t_crgi_gen_status.percent = percent_complete
| | | | | |  use msgsnd() to send the message to ssgm
| | | | | |  use f_sutl_elog() to log "start generation requested with run
| | | | | |  in progress"
| | | | |  ENDIF
| |  <-------break
| | | |
| | |  CASE c_crgi_abort_gencmd
| | | |
| | | |  IF t_crgi_RSS_state is s_sgen_idle THEN
| | | | | |  t_crgi_genrsp.abort_status = c_crgi_abort_compl
| | | | | |  t_crgi_genrsp.rsp_type = c_crgi_abort_genrsp
| | | | | |  use msgsnd() to send the message to ssgm
| | | | |  ELSE
| | | | | |  t_sgen_run_state = s_sgen_aborting
| | | | | |  signal (srun | sbit | sbmt) to terminate processing
| | | | |  ENDIF
| |  <-------break
| | | |
| | |  CASE c_crgi_shutdwn_gencmd
| | | | |  IF t_crgi_RSS_state = s_sgen_idle AND
| | | | |  t_sgen_run_state = s_sgen_idle THEN
| | | | | |  use f_sgen_terminate() to bring down the task
| | | | | |
| | | | |  ELSE
| | | | | |  t_sgen_run_state = c_sgen_terminating
| | | | | |  signal (srun | sbit | sbmt) to terminate processing
| | | | |  ENDIF
| |  <-------break
| | | |
| |  ENDSWITCH
|  ENDIF
|
|  SWITCH on t_sgen_active_status
| |
| |  CASE of c_sgen_srun_finished
| | |
| | |  SWITCH on t_sgen_run_state
| | | |
| | | |  CASE c_sgen_running
| | | | |  t_sgen_active_status = c_sgen_sbit_working
| | | | |  use fork() to start sbit
| | |  <-------break
| | | | |
| | | |  CASE c_sgen_aborting
| | | | |  use f_sgen_abort() to reset the task
| | |  <-------break
| | | | |
| | | |  CASE c_sgen_terminating
| | | | |  use f_sgen_terminate() to bring down  the task
| | | | |
| | |  ENDSWITCH
| <-------break
| | |
| | CASE of c_sbit_finished
| | |
| | | SWITCH on t_sgen_run_state
| | | |
| | | | CASE c_sgen_running
| | | | | t_sgen_active_status = c_sgen_sbmt_working
| | | | | use fork() to start sbmt()
| | | <-------break
| | | | |
| | | | CASE c_sgen_aborting
| | | | | use f_sgen_abort() to reset the task
| | | <-------break
| | | | |
| | | | CASE c_sgen_terminating
| | | | | use f_sgen_terminate() to bring down the task
| | | | |
| | | ENDSWITCH
| <-------break
| | |
| | CASE of c_sbmt_finished
| | |
| | | compute new_percent, the percentage of the generation run
| | | that is now complete, using t_sgen_number_of_cpis,
| | | t_sgen_cpi_counter
| | |
| | | SWITCH on t_sgen_run_state
| | | |
| | | | CASE c_sgen_running
| | | | | IF new_percent = 100 THEN
```

```
    | | | | | | use f_sutl_elog() to log "signal generation complete"
    | | | | | | t_crgi_RSS_state = c_sgen_idle
    | | | | | | t_sgen_run_state = c_sgen_idle
    | | | | | | use f_sutl_clog() to close the error log
    | | | | | | t_crgi_genrsp.rsp_type = c_crgi_start_genrsp
    | | | | | | t_crgi_gen_status.status = c_crgi_gen_compl
    | | | | | | use msgsnd() to send the message to ssgm
    | | | | | ELSE
    | | | | | | t_sgen_active_status = c_sgen_sbit_working
    | | | | | | use fork() to start sbit()
    | | | | | | IF (new_percent - t_sgen_percent_complete > 5) THEN
    | | | | | | | t_crgi_genrsp.rsp_type = c_crgi_start_genrsp
    | | | | | | | t_crgi_gen_status.status = c_crgi_in_progress
    | | | | | | | t_sgen_percent_complete = new_percent
    | | | | | | | t_crgi_gen_status.percent=t_sgen_percent_complete
    | | | | | | | use msgsnd() to send the message to ssgm
    | | | | | | ENDIF
    | | | | | ENDIF
    | | | <-------break
    | | | | |
    | | | | CASE c_sgen_aborting
    | | | | | use f_sgen_abort() to reset the task
    | | | <-------break
    | | | | |
    | | | | CASE c_sgen_terminating
    | | | | | use f_sgen_terminate() to bring down the task
    | | | | |
    | | | ENDSWITCH
    | <-------break
    | | |
    | | CASE of c_srun_terminated
    | | CASE of c_sbit_terminated
    | | CASE of c_sbmt_terminated
    | | | use f_sgen_failed() to do task clean-up and return the task to
    | | | and idle state
    | <-------break
    | | |
    | | CASE else
    | <-------break
    | | |
    | ENDSWITCH
    |
    DOWHILE forever
    ENDDO

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */
```

```
}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:     sgen_abort.c

Identification: [@(#)]

SCCS Id:       @(#)sgen_abort  1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:      Marg Baskin

Change History:

SCR   Date         Author  Change made
---   ----------   -----   ------------------------------------
n/a   08-APR-1993  mdb     Initial Release

--------------------------------------------------------------------

CSU_NAME:      sgen (Signal Generation Controller)

FUNCTION:      f_sgen_abort

PURPOSE:       This function does final clean-up functions needed when the
task is being aborted by a Abort Signal Generation directive
from the terminal controller.

==========================================================================*/

void f_sgen_abort ()
    /*% f_sgen_abort function  */

{

/*|
    use f_sgen_cleanup() to perform task clean-up
    set t_crgi_RSS_state = c_sgen_idle
    set t_sgen_run_state = c_sgen_idle
    set t_sgen_active_status = c_sgen_status_idle
    use f_sutl_elog() to log "signal generation aborted"
    use f_sutl_clog() to close the error log
    set t_crgi_genrsp.rsp_type = c_crgi_abort_genrsp
    set t_crgi_genrsp.abort_status = c_crgi_abort_compl
    use msgsnd() to send the message to ssgm

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==== ================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sgen_initialize.c

Identification: [@(#)]

SCCS Id:        @(#)sgen_initialize     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author  Change made
---   ----------- ------  --------------------------------------
n/a   08-APR-1993  mdb    Initial Release

-----------------------------------------------------------------------

CSU_NAME:       sgen (Signal Generation Controller)

FUNCTION:       f_sgen_initialize

PURPOSE:        This function performs all the initialize functions
required to start sgen (attaching to the message queue,
state initialization, etc.)

INPUT:          from global data: RSS_state (shared with ssgm)

OUTPUT:         to global data: active_status (shared with srun, sbit, sbmt)


=====================================================================*/

void f_sgen_initialize ()
    /*% f_sgen_initialize function */

{

/*|

    use shmget() to get shared memory key for t_crgi_RSS_state
    use shmat() to attach it
    use shmget() to get shared memory key for t_sgen_active_status
    use shmat() to attach it
    set t_sgen_run_state = s_sgen_idle
    set t_sgen_active_status = s_sgen_status_idle
    set t_crgi_RSS_state = s_sgen_idle
    use msgget() to attach to the ssgm message queue
    IF return status indicates a failure THEN
<-------return (s_spd_failure)
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*===========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sgen_failed.c

Identification: [@(#)]

SCCS Id:        @(#)sgen_failed 1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   -----------------------------------
n/a   08-APR-1993  mdb     Initial Release

---------------------------------------------------------------------

CSU_NAME:       sgen (Signal Generation Controller)

FUNCTION:       f_sgen_failed

PURPOSE:        This function does final clean-up functions needed when the
task is in an unrecoverable error state.  Each of the
subordinate tasks do their own resource clean-up before
terminating, so this task does not need to deallocate
the global database or delete files.

===========================================================================*/

void f_sgen_failed ()
    /*% f_sgen_failed function  */

{

/*|

    use f_sgen_cleanup() to perform task clean-up
    set t_crgi_RSS_state = s_sgen_idle
    set t_sgen_run_state = s_sgen_idle
    set t_sgen_active_status = s_sgen_status_idle
    use f_sutl_clog() to close the error log
    set t_crgi_genrsp.rsp_type = c_crgi_start_genrsp
    set t_crgi_gen_status.status = c_crgi_gen_failed
    use msgsnd() to send the message to ssgm

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sgen_terminate.c

Identification: [@(#)]

SCCS Id:        @(#)sgen_terminate      1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  -----------  ------  ---------------------------------------
n/a  08-APR-1993   mdb     Initial Release

----------------------------------------------------------------------------

CSU_NAME:       sgen (Signal Generation Controller)

FUNCTION:       f_sgen_terminate

PURPOSE:        This function does final clean-up functions needed when the
task is being shutdown by a Shutdown Signal Generation
directive from the terminal controller.

==========================================================================*/

void f_sgen_terminate ()
    /*% f_sgen_terminate function  */

{

/*|
    use f_sgen_cleanup() to perform task clean-up
    use f_sutl_elog() to log "Signal generation terminated"
    use f_sutl_clog() to close the error log
    delete the message queue
    use shmdt() to detach the status words (t_sgen_RSS_state and
    t_sgen_active_status)
    exit() the task

 */

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

E-81

```
==========================================================================
================
SUBJECT
    PAGE NO.   LINE NO.      ENCLOSING FUNCTION

   NO MARKS TO REFERENCE
```

INVOCATION TREE

```
PAGE NO.   LINE NO.

   1          55      sgen_main
   5         372      |  f_sgen_initialize
   7         537      |  f_sgen_terminate
   4         287      |  f_sgen_abort
   6         457      |  f_sgen_failed
```

ERROR SUMMARY

==

=================

PAGE NO. LINE NO. ERROR TEXT

NO ERRORS TO REPORT

```
                                              TBD CROSS REFERENCE REPORT
==========================================================================================================================
===============================
PAGE NO.   LINE NO.          ENCLOSING FUNCTION

  NO TBDs TO REFERENCE
```

THIS PAGE INTENTIONALLY LEFT BLANK

E4 Run Initialization (SRUN) CSU

THIS PAGE INTENTIONALLY LEFT BLANK

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□                    □□□□
□□□□                    □□□□
□□□□                    □□□□
 □□                      □□
 □□        CDADL PROCESSOR        □□
 □□                      □□
 □□                      □□
 □□         VERSION 2.4.6        □□
 □□                      □□
 □□     SERIAL NUMBER: C38U0011        □□
 □□                      □□
 □□        AUTHORIZED USER:        □□
 □□        Raytheon - Canada        □□
 □□                      □□
 □□                      □□
 □□                      □□
 □□                      □□
 □□                      □□
□□□□                    □□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

1 FILE srun.c
2 FILE srun_creat_srsd.c
3 FILE srun_det_patch_valid.c
4 FILE srun_cleanup.c
5 FILE srun_ec_map.c
7 FILE srun_filter_store.c
8 FILE srun_gen_seed.c
9 FILE srun_iff.c
10 FILE srun_init_ssp.c
11 FILE srun_initialize.c
13 FILE srun_load_trdb.c
14 FILE srun_mercator.c
16 FILE srun_rx_db.c
18 FILE srun_seeds.c
19 FILE srun_straddle_check.c
20 FILE srun_straddle_points.c

21 FILE srun_orbit_coeffs.c
22 FILE srun_getparams.c
23 REPORT Requirements Traceability
24 REPORT INVOCATION TREE
25 REPORT ERROR SUMMARY

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun.c

Identification: [@(#)]

SCCS Id:        @(#)srun        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------  ------  ---------------------------------
n/a  08-APR-1993  mdb     Initial Release

---------------------------------------------------------------------

CSU_NAME:       srun (Initialize Run)

FUNCTION:       srun_main

PURPOSE:        This function localizes all generation activities which are
performed once at the beginning of a generation run.

This task is activated by a fork()/execcv() within sgen.
It takes all its parameters from shared global data, and builds
the Run Init Constants database in shared global data for
access by sbit, sbmt and (status only) sgen.

INPUT:          --

OUTPUT:         to shared data:
sub-structure elements of t_sgen_run_constants
sub-structure elements of t_sgen_run_parameters
t_sgen_active_status
t_crad_sat_def (in memory copy)
t_cend_env_def (in memory copy)

EXCEPTIONS:   Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

#include "srun_priv.h"

void srun_main ()
    /*% srun function */

```

```
{

/*|
    use f_srun_initialize() to set up the run variables
    use f_srun_orbit_coeffs() to calculate the orbital model coefficients
    IF (t_cend_atmosphere.enabled = c_spd_true) THEN
    |   use f_srun_load_trdb() to load the tropospheric refraction database
    ENDIF
    use f_srun_rx_db() to set up the front end database
    use f_srun_ec_map() to generate the earth clutter map
    set t_sgen_active_status = c_sgen_srun_finished

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=================================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_creat_srsd.c

Identification: [@(#)]

SCCS Id:        @(#)srun_creat_srsd     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- -----    -----------------------------------
n/a   08-APR-1993  mdb     Initial Release

------------------------------------------------------------------

CSU_NAME:        srun (Initialize Run)

FUNCTION:        f_srun_creat_srsd

PURPOSE:         This function creates the SRSD file for the current run,
and initializes its header block structure.  This file
is added to on a CPI-by-CPI basis, as the samples are
generated, and the header block information updated with
final information prior to the termination of the run.

INPUT:           from shared data:
output_dir: directory specification to use
t_sgen_srsd_header: SRSD header block structure
gen_ident: user ID used to build the output filename

OUTPUT:          to shared data:
srsd_hndl: SRSD file handle

EXCEPTIONS:      Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

=================================================================================*/

void f_srun_creat_srsd ()
    /*% f_srun_creat_srsd function */

{

/*|

    open SRSD file in directory <output_dir>, filename = "<gen_ident>.sg.srsd"
    with "write"
    store t_sgen_srsd_header to the file

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_det_patch_valid.c

Identification: [@(#)]

SCCS Id:        @(#)srun_det_patch_valid        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:      Marg Baskin

Change History:

SCR    Date        Author    Change made
---  ----------  -----  ---------------------------------
n/a  08-APR-1993   mdb     Initial Release

---------------------------------------------------------------------

CSU_NAME:        srun (initialize run)

FUNCTION:        f_srun_det_patch_valid

PURPOSE:         This function is called once per earth clutter patch (as
defined in the HMI), to determine if the patch is "valid",
ie, if it can be modelled by the algorithm.

It also translates the longitude of each vertex from the
input range (-PI to PI) to range needed for processing
(0 to 2PI).

INPUTS:          patch: the patch structure for the current patch:

OUTPUTS:         TRUE, if patch can be modelled.
loc.lon has PI added to its original value

FALSE, if patch cannot be modelled (in which case is it
discarded from the earth clutter db)

EXCEPTIONS:    Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_srun_det_patch_valid ()
    /* t_spd_boolean f_srun_det_patch_valid (t_cend_clutter_patch patch) */
    /*% f_srun_det_patch_valid function */

{

```

```
/*|

    IF patch can be modelled THEN
<-------return(c_spd_true)
    ELSE
<-------return(c_spd_false)
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_cleanup.c

Identification: [@(#)]

SCCS Id:        @(#)srun_cleanup        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------- -----  -------------------------------
n/a  08-APR-1993  mdb     Initial Release

--------------------------------------------------------------------

CSU_NAME:       srun (Initialize Run)

FUNCTION:       i_srun_cleanup

PURPOSE:        This function performs clean-up activities prior to the
termination of srun.  This includes deallocation of
memory allocated to the global database, closing and
clean-up of intermediate files, etc.

It is set up as an interrupt handler, and is entered
by srun and and any of its subfunctions upon recognition of an
unrecoverable error condition.

EFFECTS:    Sets t_sgen_active_status = c_sgen_sbrun_terminated

==========================================================================*/

void i_srun_cleanup ()
    /*% i_srun_cleanup function */

{

/*|

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_ec_map.c

Identification: [@(#)]

SCCS Id:        @(#)srun_ec_map 1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR    Date        Author    Change made
---  -----------  ------  ---------------------------------------
n/a  08-APR-1993   mdb     Initial Release

------------------------------------------------------------------------

CSU_NAME:       srun (initialize run)

FUNCTION:       f_srun_ec_map

PURPOSE:        This function is called once per signal generation run, to
generate the Earth Clutter map for the run.

INPUT:          From shared data:
Earth Clutter patch descriptions (from cend):
patch_count (number of clutter patches)
patch (clutter patch structure, one per defined patch)

OUTPUT:         To shared data:
npplat (number of patches per latitude line)
clutter_map_db (clutter map database)

EXCEPTIONS:     Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

========================================================================*/


void f_srun_ec_map ()
    /*% f_srun_ec_map function */

{

/*|
    FOR each patch (t_cend_env_def.patch[ip] for ip = 1 to
    |   t_cend_env_def.patch_count)
    LOOP
```

```
| IF (f_srun_det_patch_valid (patch[ip]) = c_spd_true) THEN
| | IF (f_srun_mercator(patch[ip], npf, subpatch) = c_spd_true) THEN
| | | FOR each subpatch (1 to npf)
| | | LOOP
| | | | Compute minlat & maxlat, the patch's latitude limits
| | | | FOR each minute of latitude, i, from minlat to maxlat
| | | | LOOP
| | | | | Increment t_sgen_run_constants.npplat[i],
| | | | | the number of patch borders along the latitude line
| | | | | Compute the minlon, maxlon, the longitude extents at this
| | | | | minutes of latitude
| | | | | set t_sgen_clutter_map_db.lon_min[i][npplat[i]] = minlon
| | | | | set t_sgen_clutter_map_db.lon_max[i][npplat[i]] = maxlon
| | | | | set t_sgen_clutter_map_db.patch_idx[i][npplat[i]] = ip
| | | | | Increment t_sgen_run_constants.npplat[i],
| | | | | the number of patch borders along the latitude line
| | | | ENDLOOP by latitude line
| | | ENDLOOP by subpatch
| | ELSE
| | | the patch cannot be modelled
| | | use f_sutl_elog() to log "invalid earth clutter patch, patch #<ip>"
| | ENDIF
| ELSE
| | the patch cannot be modelled
| | use f_sutl_elog() to log "invalid earth clutter patch, patch #<ip>"
| ENDIF
ENDLOOP by patch
optimize t_sgen_clutter_map_db for later usage, by shuffling the elements
along each latline into ascending order by lon_min

*/
```

```

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_filter_store.c

Identification: [@(#)]

SCCS Id:        @(#)srun_filter_store   1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------  ------  ---------------------------------
n/a  08-APR-1993  mdb     Initial Release

-------------------------------------------------------------------------

CSU_NAME:       srun (Initialize Run)

FUNCTION:       f_srun_filter_store

PURPOSE:        This function stores the IF Filter Response vector for
selected antennas.

INPUT:          from global data: t_crad_sat_def

OUTPUT:         An XDR-encoded output file containing IF Filter Response data.
The file is stored in directory <output_dir>, with a filename
of "<gen_ident>.ir.rss_filt_rsp"

EXCEPTIONS:     Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

=========================================================================*/

void f_srun_filter_store ()
    /*% f_srun_filter_store function */

{

/*|

    initialize t_cgep_if_filter_header
    open the file and create an XDR encode output stream
    put t_cgep_if_filter_header to the file
    FOR each channel (1 to number_of_channels)
    LOOP
    |  IF the instance of data_capture.cap_filt_rsp associated with that
    |  channel = c_spd_true THEN
    |  |  put the t_sgen_if_filter_response row for that channel to
    |  |  t_cgep_if_filter_chan, and append to the XDR external file
    |  ENDIF
    ENDLOOP by channel
    close the file and release the XDR datastream

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*============================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_gen_seed.c

Identification: [@(#)]

SCCS Id:        @(#)srun_gen_seed       1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date         Author   Change made
---   -----------  ------   ----------------------------------------
n/a   08-APR-1993  mdb      Initial Release

-----------------------------------------------------------------------------

CSU_NAME:       srun (Initialize Run)

FUNCTION:       f_srun_gen_seed

PURPOSE:        This function is used to generate a starting random seed
vector for any generation sequence.

The seed numbers returned by this routine are intended to be
repeatable, so that a subsequent generation run with the same
HMI-specified parameters will produce predictable results.

INPUT:          a 32-bit integer used as a seed

OUTPUT:         a vector of 97 numbers, to be used as a seed vector

EXCEPTIONS:     Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

============================================================================*/

void f_srun_gen_seed ()
    /* void f_srun_gen_seed (
    t_spd_int32 seed,
    t_sgen_seed vector[97]) */
    /*% f_srun_gen_seed function */

{

/*|

    generate vector, a sequence of 97 floating-point numbers, starting with seed
    using the algorithm specified in "Numerical Recipes", pg 196-7

 */

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=======================================================================
SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_iff.c

Identification: [@(#)]

SCCS Id:        @(#)srun_iff    1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date         Author   Change made
---  -----------  ------  ---------------------------------------
n/a  08-APR-1993   mdb      Initial Release

-------------------------------------------------------------------------

CSU_NAME:       srun (initialize run)

FUNCTION:       f_srun_iff

PURPOSE:        This function generates the if_filter_response table for
the current run.

INPUT:          from global data:
t_crad_sat_def

OUTPUT:         to global data:
t_sgen_run_constants.if_filter_response

EXCEPTIONS:     Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

=======================================================================*/

void f_srun_iff ()
    /*% f_srun_iff function */

{

/*|

    compute if_filter_response

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_init_ssp.c

Identification: [@(#)]

SCCS Id:        @(#)srun_init_ssp       1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------- ------  ------------------------------------
n/a  08-APR-1993  mdb     Initial Release

--------------------------------------------------------------------------

CSU_NAME:       srun (Initialize run)

FUNCTION:       f_srun_init_ssp

PURPOSE:        This function computes the initial SSP latitude and
SSP longitude for the run, if the Sat/Rad definition
provided this in terms of relative angles.

INPUT:          from t_crad_sat_def
:
orbital_pos.incl_angle
orbital_pos.direction
orbital_pos.ref_loc

OUTPUT:         to global data (t_crad_sat_def):
orbital_pos.SSP_loc

EXCEPTIONS:   Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

A fatal error occurs if the viewing geometry is inconsistant
(which should never happen, if these values have been
correctly validated within the HMI).

==========================================================================*/

void f_srun_init_ssp ()
    /*% f_srun_init_ssp function */

{

```

```
/*!

    compute SSP_loc.lat
    compute SSP_loc.lon

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=====================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_initialize.c

Identification: [@(#)]

SCCS Id:        @(#)srun_initialize     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author  Change made
---  ----------- ------  ----------------------------------------
n/a  08-APR-1993  mdb     Initial Release

-------------------------------------------------------------------------

CSU_NAME:       srun (Initialize run)

FUNCTION:       f_srun_initialize

PURPOSE:        This function localizes all the initialization, filtering
and working-parameter set-up needed for a generation run.
It builds its data into the global database which is accessed
by sbit and sbmt throughout the course of the run.

INPUT:          --

OUTPUT:         t_crad_sat_def
t_cend_env_def
portions of t_sgen_run_constants; t_sgen_run_parameters

EXCEPTIONS:     Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

=====================================================================*/

void f_srun_initialize ()
    /*% f_srun_initialize function */

{

/*|
    Use f_srun_getparams() to load the Signal Generation Parameters
    from the files pointed to by satdef_pathname and envdef_pathname and
    filter Signal Generation Parameters into the required units, as
    necessary

```

```
In t_sgen_run_parameters, initialize current_dwell, current_nci, current_cpi,
cpi_counter

Compute number_of_cpis, the total number of CPIs in the run

IF t_crad_sat_def.orbital_pos.loc_method != c_spd_direct_loc_method THEN
|   Use f_srun_init_ssp() to compute t_crad_sat_def.orbital_pos.SSP_loc
ENDIF

Set t_sgen_cpi_database.SSP_current = SSP_loc

Set t_sgen_cpi_database.current_direction to
t_crad_sat_def.orbital_pos.direction

Compute orbit_radius, rotational_velocity, k1, k3, velocity,
number_of_antennas, number_of_channels, range_quant, slant_range,
num_rs, replica_interpolation_factor, elevation_angles, iratio, out_off

Initialize trans_start_time, trans_end_time, receive_start_time_ref,
receive_end_time_ref

Use f_srun_seeds() to initialize the random number sequences

Compute noise_window

Compute first_channel, the array specifying the first channel number
associated with each antenna

Generate nonlinearity_params, the table of nonlinearity parameters for the
antennas
```

```
    Compute antenna random sequence lengths and generate RandU and RandV, the
    Gaussian white noise samples for the antenna patterns

    Get the amount of free_space available for intermediate product storage

    FOR each of the possible intermediate products streams
    LOOP
    |  IF the intermediate product is needed THEN
    |  |  Estimate the output product size & subtract from free_space
    |  |  IF space is available THEN
    |  |  |  use f_cget_write() to create the corresponding product file
    |  |  ELSE
    |  |  |  Use sutl_elog() to log "insufficient disk space for intermediate
    |  |  |  products"
    |  |  |  Trap to i_srun_cleanup()
    |  |  ENDIF
    |  ENDIF
    ENDLOOP by product stream

    Get the amount of free_space available for Signal Processing Parameters
    Estimate the output product size
    IF space is available THEN
    |  Use f_cget_write() to create the Signal Processing Parameters file
    ELSE
    |  Use sutl_elog() to log "insufficient disk space for Signal Processing
    |  Parameters file"
    |  Trap to i_srun_cleanup()
    ENDIF

    Get the amount of free_space available for SRSD
    Estimate the output product size
    IF space is available THEN
    |  Use f_srun_creat_srsd() to create the SRSD output file
    ELSE
    |  Use sutl_elog() to log "insufficient disk space for SRSD file"
    |  Trap to i_srun_cleanup()
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_load_trdb.c

Identification: [@(#)]

SCCS Id:        @(#)srun_load_trdb      1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------  -----    -------------------------------
n/a   08-APR-1993  mdb     Initial Release

--------------------------------------------------------------------------

CSU_NAME:       srun (initialize run)

FUNCTION:       f_srun_load_trdb

PURPOSE:        This function loads a portion of the tropospheric refraction
database, based on the value of humidity.  It is called
only if Lower Atmosphere is enabled for the run.

INPUT:          from global data:
t_cend_env_def.ionsph.humid
from an external UNIX file: data to load into trop_ref_db

OUTPUT:         to global data:
t_sgen_run_constants.trop_ref_db

EXCEPTIONS:     Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_srun_load_trdb ()
    /*% f_srun_load_trdb function */

{

/*|
    IF t_cend_env.atmos.humid = c_spd_CCIR_avrg THEN
    |  load the CCIR average file from disk into trop_ref_db
    ELSE
    |  load the 100% humidity file from disk into trop_ref_db
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_mercator.c

Identification: [@(#)]

SCCS Id:        @(#)srun_mercator      1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------  -----    ------------------------------
n/a   08-APR-1993  mdb     Initial Release

---------------------------------------------------------------------------

CSU_NAME:       srun (initialize run)

FUNCTION:       f_srun_mercator

PURPOSE:        This function is called once per valid earth clutter patch,
to determine if the patch straddles the mercator edge, and
break it up into multiple patches (none of them straddling)
if necessary.

INPUT:          patch: patch description

OUTPUT:         TRUE, if the operation was successful, plus
npf - number of sub-patches
subpatch (one per subpatch, to a max of 2).  If the patch
was not broken up, then the first subpatch is the
original patch.

FALSE, if the operation failed, in which case the patch is
discarded from the earth clutter db.

EXCEPTIONS:     Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

=========================================================================*/

void f_srun_mercator ()
   /* t_spd_boolean f_srun_mercator (
   t_cend_clutter_patch patch,
   t_spd_card8 npf,
   t_cend_clutter_patch subpatch[2]) */
   /*% f_srun_mercator function */
```

E-1.5

```

{

/*!

    build subpatch[0] = patch
    set stradcount = 0
    FOR each vertex pair (loc.lon[i], loc.lon[i+1]) defined for the patch
    LOOP
    |  IF (f_srun_straddle_check(loc.lon[i], loc.lon[i+1]) = c_spd_true) THEN
    |  |  increment stradcount
    |  |  set straddle(stradcount) = i
    |  ENDIF
    ENDLOOP
    IF stradcount = 2, ie, if the patch crosses the line THEN
    |  use f_srun_straddle_points(straddle, patch.loc) to compute
    |  the crossing points
    <---break the patch down into two subpatches, and store them in subpatch[0]
    |  and subpatch[1]
    ELSEIF stradcount != 0 THEN
<-------return (c_spd_false), and discard the patch from further processing
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=============================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_rx_db.c

Identification: [@(#)]

SCCS Id:        @(#)srun_rx_db  1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------  ------  -----------------------------------
n/a  08-APR-1993  mdb     Initial Release

-----------------------------------------------------------------------

CSU_NAME:       srun (Initialize Run)

FUNCTION:       f_srun_rx_db

PURPOSE:        This function sets up the front end related elements of the
shared global database at the beginning of a generation run.

INPUT:          from global data:
t_crad_sat_def.antenna_model.main_rxfe
data_capture struct from each antenna
t_sgen_run_constants
t_sgen_run_parameters

OUTPUT:         to global data:
t_sgen_run_constants.if_filter_response
.omega_if
.iq_parameters
.ad_parameters

EXCEPTIONS:   Whenever a fatal error is encountered, i_srun_cleanup() is
executed, and further processing is abandoned.

=============================================================================*/

void f_srun_rx_db ()
    /*% f_srun_rx_db function */

{

/*|
     use f_srun_iff() to compute if_filter_response

     use f_srun_filter_store() to store if_filter_response in the
     intermediate product file

     Compute omega_if, the A/D IF angular frequency

     FOR each antenna (1 to number_of_antennas)
     LOOP
     |  Compute iq_parameters
     ENDLOOP by channel

     FOR each channel (1 to number_of_channels)
     LOOP
     |  Compute ad_parameters
     ENDLOOP by channel

*/

     /* Local macros */

     /* Local constants */

     /* Local Structure Definitions */

     /* Local Variables */

     /* Output Messages */

     /* Code */

```

```
1114 }
```

E-120

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_seeds.c

Identification: [@(#)]

SCCS Id:        @(#)srun_seeds  1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ---------  ----  ------------------------------
n/a  08-APR-1993   mdb     Initial Release

---------------------------------------------------------------------

CSU_NAME:       srun (Initialize Run)

FUNCTION:       f_srun_seeds

PURPOSE:        This function generates the starting random seed vectors
for all generation sequences.

The seed numbers returned by this routine are intended to be
repeatable, so that a subsequent generation run with the same
HMI-specified parameters will produce predictable results.

INPUT:          from global data:
t_cend_env_def: all seed fields (ec_seed, vc_seed, etc.)

OUTPUT:         to global data:
t_sgen_run_parameters.all_seeds

EXCEPTIONS:   Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_srun_seeds ()
    /*% f_srun_seeds function */

{

/*|

    FOR each seed in t_crad_env_def (noise, ionosphere, earth clutter,
    |   discretes, volumetric, jammers, targets)
    LOOP
    |  IF RSS_spec for the seed = c_spd_true THEN
    |  |  use f_srun_gen_seed() with corresponding seed constant
    |  |  (eg. c_sgen_ec_seed for earth clutter seed)
    |  ELSE
    |  |  use f_srun_gen_seed() with seed specified in t_crad_env_def
    |  ENDIF
    ENDLOOP

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*======================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_straddle_check.c

Identification: [@(#)]

SCCS Id:        @(#)srun_straddle_check 1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------  ------   ----------------------------------
n/a   08-APR-1993  mdb     Initial Release

----------------------------------------------------------------------

CSU_NAME:       srun (initialize run)

FUNCTION:       f_srun_straddle_check

PURPOSE:        This function is called by f_srun_mercator once per vertex
pair, to test whether the side connecting the vertices crosses
the mercator edge.

INPUT:          longstart: starting longitude of side
longend: ending longitude of side

OUTPUT:         TRUE, if the side straddles the edge

FALSE, if the side does not straddle the edge

EXCEPTIONS:     Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

======================================================================*/

void f_srun_straddle_check ()
    /* t_spd_boolean f_srun_straddle_check (
    t_spd_lon longstart,
    t_spd_lon longend) */
    /*% f_srun_straddle_check function */

{

/*!

    IF the side drawn between longstart and longend straddles the mercator edge,
    THEN  return (c_spd_true)
    ELSE  return (c_spd_false)
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*============================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_straddle_points.c

Identification: [@(#)]

SCCS Id:        @(#)srun_straddle_points        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ------------------------------------
n/a   08-APR-1993  mdb     Initial Release

------------------------------------------------------------------------------

CSU_NAME:       srun (initialize run)

FUNCTION:       f_srun_straddle_points

PURPOSE:        This function is called by f_srun_mercator to compute the
points at which the sides cross the mercator edge.

INPUT:          straddle: starting vertices of straddling sides
vert: vertex vector for patch

OUTPUT:         TRUE, if the calculation succeeds
spoint[2][2]: the lat/lon pairs of the intersection points

FALSE, if the calculation fails

EXCEPTIONS:     Whenever a fatal error is encountered, i_srun_cleanup() is
executed, and further processing is abandoned.

============================================================================*/

void f_srun_straddle_points()
    /* t_spd_boolean f_srun_straddle_points (
    t_spd_card8 straddle,
    t_spd_location vert[c_cend_max_vertices],
    t_spd_location spoint[2]) */
    /*% f_srun_straddle_points function */

{

/*|

    FOR each of the two crossing sides
    LOOP
    |  compute spoint.lat and spoint.lon for the intersection point
    |  with the mercator edge
    ENDLOOP

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_orbit_coeffs.c

Identification: [@(#)]

SCCS Id:        @(#)srun_orbit_coeffs   1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date          Author   Change made
---   -----------   -----    ---------------------------------
n/a   08-APR-1993   mdb      Initial Release

--------------------------------------------------------------------------

CSU_NAME:       srun (initialize run)

FUNCTION:       f_srun_orbit_coeffs

PURPOSE:        This function is called once per generation run.  It
calculates the baseline orbital model coefficients for
the run and stores them in the shared global database.

INPUTS:         from global data:
t_crad_sat_def.orbital_pos

OUTPUTS:        to global data:
t_sgen_run_constants.orbital_model_coeffs

EXCEPTIONS:     Whenever a fatal error is encountered, i_srun_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_srun_orbit_coeffs ()
    /*% f_srun_orbit_coeffs function */
{

/*|

    IF incl_angle = 0 THEN
    |  compute time_zero, omega_zero using equitorial orbit algorithm
    ELSEIF incl_angle = PI/2 THEN
    |  compute time_zero, omega_zero using polar orbit algorithm
    ELSE
    |  compute time_zero, omega_zero using general algorithm
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Codes */

}
```

```
/*============================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      srun_getparams.c

Identification: [@(#)]

SCCS Id:        @(#)srun_getparams      1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR    Date         Author    Change made
---  -----------  ------  ------------------------------------
n/a  08-APR-1993    mdb       Initial Release

------------------------------------------------------------------------

CSU_NAME:       srun (Initialize Run)

FUNCTION:       f_srun_getparams

PURPOSE:        This function reads the two files which make up the
Signal Generation Parameters (Satellite/Antenna &
Environment) into in-memory data structures.
These files are stored in xdr format and accessed via
xdr routines.

Once the files are loaded, they are filtered on a field-by-
field basis to convert them into the units of measure
needed by the RSS simulation (degrees to radians, metres to
kilometres, etc.)

In those cases where the starting values of a field are
required by the simulation, those starting values are kept
as working variables.  The remainder of the original
values are lost after the conversation process, in order to
save memory.

EXCEPTIONS:     Whenever an error is encountered, i_srun_cleanup() is executed
and further processing is abandoned.

============================================================================*/

void f_srun_getparams ()
    /* void f_srun_getparams (
    char *satdef_pathname,
    char *envdef_pathname) */
    /*% f_srun_getparams function */
```

```

{

/*|

    use f_crad_read() to get the Sat/Ant definitions into memory
    use f_cend_read() to get the Environment definitions into memory

    FOR each field in the two structures
    LOOP
    |  filter the field into the units of measure needed by the RSS generation
    |  The enumeration of specific translations exists as a modified HMI
    |  table.  In general, degrees -> radians, kilometres -> metres, etc.
    ENDLOOP

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

==

================

SUBJECT

PAGE NO.   LINE NO.   ENCLOSING FUNCTION

NO MARKS TO REFERENCE

E-131

INVOCATION TREE

```
===============================================================================
================
PAGE NO.   LINE NO.


   1         56      srun_main
  11        760       | f_srun_initialize
  22       1481       | | f_srun_getparams
  10        688       | | f_srun_init_ssp
  18       1164       | | f_srun_seeds
   8        539       | | | f_srun_gen_seed
   4        287       | | i_srun_cleanup
   2        136       | | f_srun_creat_srsd
  21       1399       | f_srun_orbit_coeffs
  13        902       | f_srun_load_trdb
  16       1076       | f_srun_rx_db
   9        611       | | f_srun_iff
   7        457       | | f_srun_filter_store
   5        358       | f_srun_ec_map
   3        214       | | f_srun_det_patch_valid
  14        981       | | f_srun_mercator
  19       1244       | | | f_srun_straddle_check
  20       1321       | | | f_srun_straddle_points
```

ERROR SUMMARY

PAGE NO. LINE NO. ERROR TEXT

NO ERRORS TO REPORT

```
                                        TBD CROSS REFERENCE REPORT
======================================================================================================
===========================
PAGE NO.   LINE NO.          ENCLOSING FUNCTION

  NO TBDs TO REFERENCE
```

E5 Burst Initialization (SBIT) CSU

THIS PAGE INTENTIONALLY LEFT BLANK

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□                    □□□□
□□□□                    □□□□
□□□□                    □□□□
 □□                      □□
 □□       CDADL PROCESSOR       □□
 □□                      □□
 □□                      □□
 □□        VERSION 2.4.6       □□
 □□                      □□
 □□    SERIAL NUMBER: C38U0011      □□
 □□                      □□
 □□       AUTHORIZED USER:       □□
 □□       Raytheon - Canada       □□
 □□                      □□
 □□                      □□
 □□                      □□
 □□                      □□
 □□                      □□
□□□□                    □□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

1 FILE sbit.c
3 FILE sbit_initialize.c
4 FILE sbit_cleanup.c
5 FILE sbit_atten.c
6 FILE sbit_antenna.c
8 FILE sbit_beam_centre.c
10 FILE sbit_cell_valid.c
11 FILE sbit_cgep_store.c
12 FILE sbit_ionosphere.c
13 FILE sbit_jammer_db.c
15 FILE sbit_noise_db.c
16 FILE sbit_rd_map.c
18 FILE sbit_target_db.c
19 FILE sbit_chaff_db.c
20 FILE sbit_precip_db.c
22 FILE sbit_precip_valid.c

23 FILE sbit_waveform.c
25 REPORT Requirements Traceability
26 REPORT INVOCATION TREE
27 REPORT ERROR SUMMARY

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit.c

Identification: [@(#)]

SCCS Id:        @(#)sbit        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------- ------  ----------------------------------
n/a  08-APR-1993   mdb    Initial Release

--------------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       sbit_main

PURPOSE:        This function localizes all those processes which must
run at the beginning of each burst, in order to build
the burst database: generation of waveform coefficients,
attenuation database, earth clutter database, volumetric
clutter database, target database, jammer database,
antenna patterns, noise database, and ionospheric model
efficients.

This task is activated by a fork()/execcv() within sgen.
It takes all its parameters from shared global data, and
builds the burst database in shared global data, for later
access by sbmt and (status only) sgen.

INPUT:          from shared data:
t_crad_sat_def
t_cend_env_def
t_sgen_run_constants
t_sgen_run_parameters

OUTPUT:         to shared data:
substructure elements of t_sgen_burst_database

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

```

```
#include "sbit_priv.h"

void sbit_main ()
    /*% sbit_main function  */

{

/*|

    use f_sbit_initialize() to initialize the working variables
    use f_sbit_waveform() to generate the waveform coefficients
    use f_sbit_atten() to compute the attenuation database
    use f_sbit_rd_map() to generate the range-doppler & earth clutter db
    use f_sbit_precip_db() to generate the precipitation clutter database
    use f_sbit_chaff_db() to generate the chaff clutter database
    use f_sbit_target_db() to generate the target database
    use f_sbit_jammer_db() to generate the jammer database
    use f_sbit_antenna() to generate the antenna patterns
    use f_sbit_noise_db() to generate the noise database
    IF (t_cend_ionosphere.enabled = c_spd_true) THEN
    |   use f_sbit_ionosphere() to generate the ionospheric model coefficients
    ENDIF
    set t_sgen_active_status = c_sgen_sbit_finished

 */

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_initialize.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_initialize     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ----------------------------------------
n/a   08-APR-1993  mdb     Initial Release

----------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_initialize

PURPOSE:        This function computes all set-up and working parameters for
the current CPI.  It is called at the beginning of each
CPI.

INPUT:          from global data
t_crad_sat_def
t_cend_env_def
t_sgen_run_constants
t_sgen_run_parameters

OUTPUT:         to global data
substructures of t_sgen_burst_database

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_sbit_initialize ()
    /*% f_sbit_initialize function  */

{

/*|

    Increment current_cpi, cpi_counter
    compute current_direction, SSP_current, sbr_heading, k2

    IF starting a new dwell, i THEN
    |  reset current_dwell, current_nci, current_cpi
    |  IF location.loc_method for dwell[i] !=
    |  c_spd_direct_loc_method THEN
    |  |   use f_sbit_beam_centre() to compute DI[i].beam_centre.lat & .lon
    |  ENDIF
    |  set t_sgen_dwell_database.beam_centre_latitude,
    |  .beam_centre_longitude from DI[i].beam_centre
    ELSEIF starting a new nci THEN
    |  reset current_nci, current_cpi
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_cleanup.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_cleanup        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ---------------------------------------
n/a   08-APR-1993  mdb     Initial Release

---------------------------------------------------------------------

CSU_NAME:       sbit (Burst Initialization)

FUNCTION:       i_sbit_cleanup

PURPOSE:        This function performs clean-up activities prior to the
premature termination of sbit.  This includes deallocation of
memory allocated to the global database, closing and
clean-up of intermediate files, etc.

It is set up as an interrupt handler, and is entered
by sbit and and any of its subfunctions upon recognition of an
unrecoverable error condition.

EFFECTS:        sets t_sgen_active_status = c_sgen_sbit_terminated

=========================================================================*/

i_sbit_cleanup ()
    /*% i_sbit_cleanup function */

{

/*|

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*===========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_atten.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_atten  1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR    Date        Author   Change made
---  -----------  ------  -------------------------------------
n/a  08-APR-1993   mdb      Initial Release

------------------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_atten

PURPOSE:        This function computes the attenuation database for
the current CPI.

INPUT:          from shared data:
t_crad_env_def

OUTPUT:         to shared data:
t_sgen_at_attenuation
t_sgen_precip_db

EXCEPTIONS:   Whenever a fatal error is encountered,  i_sbit_cleanup()
is executed, and further processing is abandoned.

===========================================================================*/

void f_sbit_atten ()
   /*% f_sbit_atten function  */

{

/*|

   FOR each precipitation cell (ip = 1 to t_cend_env_def.cell_count)
   LOOP
   |  IF (f_sbit_cell_valid(cell[ip]) = c_spd_true) THEN
   |  |  in t_sgen_precip_db, compute cell_azimuth_extent_x,
   |  |  cell_azimuth_extent_y, cell_elevation_x, cell_elevation_y,
    | | cell_attenuation, cell_thickness
    | ENDIF
    ENDLOOP

    FOR grazing angles from 1 to 90
    LOOP
    | Compute t_sgen_at_attenuation[i] for the grazing angle
    ENDLOOP

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=============================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_antenna.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_antenna        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ---------------------------------------
n/a   08-APR-1993  mdb     Initial Release

-----------------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_antenna

PURPOSE:        This function generates the transmit antenna pattern and the
receive antenna patterns (one per antenna) for the current
CPI.  It is called once per CPI.

INPUT:          from shared data:
t_crad_sat_def

OUTPUT:         to shared data:
t_sgen_tx_ant_pattern
t_sgen_rx_ant_pattern, one per antenna

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

=============================================================================*/

void f_sbit_antenna ()
    /*% f_sbit_antenna function  */

{

/*!

    IF main antenna concept (t_spd_concepts) = c_spd_CFA THEN
    |  IF t_crad_CFA_antenna.pattern_RSS_gen = c_spd_true THEN
    |  |  generate t_sgen_gainu, t_sgen_gainv, t_sgen_egainu, t_sgen_egainv for
    |  |  the transmit antenna
|  ELSE
|  |  load t_sgen_gainu, t_sgen_gainv for the transmit antenna
|  ENDIF
|
|  FOR each DPCA group (1 to t_crad_CFA_antenna.num_DPCA-grps)
|  LOOP
|  |  FOR each phase centre within the DPCA group
|  |  |  (1 to t_crad_CFA_antenna.num_ph_ctr_per_grp)
|  |  LOOP
|  |  |  IF t_crad_CFA_antenna.pattern_RSS_gen = c_spd_true THEN
|  |  |  |  generate t_sgen_gainu, t_sgen_gainv, t_sgen_egainu,
|  |  |  |  t_sgen_egainv for the receive antenna
|  |  |  ELSE
|  |  |  |  load t_sgen_gainu, t_sgen_gainv for the receive antenna
|  |  |  ENDIF
|  |  ENDLOOP by phase centre
|  ENDLOOP by DPCA group
ENDIF

initialize sample start times for auxiliaries

IF the main antenna concept (t_spd_concepts) = c_spd_RRA THEN
|  IF t_crad_CFA_antenna.pattern_RSS_gen = c_spd_true THEN
|  |  generate t_sgen_gainu, t_sgen_gainv, t_sgen_egainu,
|  |  t_sgen_egainv for the transmit antenna
|  |  generate t_sgen_gainu, t_sgen_gainv, t_sgen_egainu,
|  |  t_sgen_egainv for the receive antenna
|  ELSE
|  |  load t_sgen_gainu, t_sgen_gainv for the transmit antenna
|  |  load t_sgen_gainu, t_sgen_gainv for the receive antenna
|  ENDIF
```

```
    ENDIF

    FOR each auxiliary antenna
    LOOP
    |  IF auxiliary type (t_spd_array_types) = c_spd_single_element THEN
    |  |  generate t_sgen_gainu, t_sgen_gainv, t_sgen_egainu,
    |  |  t_sgen_egainv for the receive antenna
    |  ENDIF
    |
    |  IF auxiliary type  (t_spd_array_types) = c_spd_regular_array THEN
    |  |  determine scanning parameters
    |  |  IF auxiliary is coplanar and within CFA aperture and has the
    |  |  same element type as the CFA THEN
    |  |  |  generate t_sgen_gainu, t_sgen_gainv, t_sgen_egainu,
    |  |  |  t_sgen_egainv for the receive antenna
    |  |  ELSE
    |  |  |  generate errors for auxiliary and use to generate pattern
    |  |  |  (method TBD)
    |  |  ENDIF
    |  ENDIF
    |
    |  IF auxiliary type  (t_spd_array_types) = c_spd_arbitrary_array THEN
    |  |  determine scanning parameters
    |  |  generate t_sgen_gainu, t_sgen_gainv, t_sgen_egainu,
    |  |  t_sgen_egainv for the receive antenna
    |  ENDIF
    |
    ENDLOOP by auxiliary antenna

    FOR each channel
    LOOP
    |  IF the corresponding instance of data_capture.capt_ant = c_spd_true THEN
    |  |  store generated antenna pattern in the intermediate
    |  |  product file
    |  ENDIF
    ENDLOOP

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

C 120

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_beam_centre.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_beam_centre    1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR  Date         Author  Change made
---  ----------- -----   --------------------------------
n/a  08-APR-1993  mdb     Initial Release

--------------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_beam_centre

PURPOSE:        This function computes the beam centre lat/lon for
the current dwell from location angles given as either
a) beam centre azimuth and grazing angles or
b) beam centre specified as vertical and horizontal
scan angle offsets from boresight. (RRA only)

It is called by f_sbit_initialize at the beginning of
each dwell.

INPUT:          from global data

OUTPUT:         to global data

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

A fatal error occurs if the geometry is inconsistant
(which should never happen, if these values have been
correctly validated within the HMI).

==========================================================================*/

void f_sbit_beam_centre ()
    /*% f_sbit_beam_centre function */

{

```

```
/*|

    IF location.loc_method for current_dwell = c_spd_azim_graz_angles_method THEN
    |  compute beam_centre_elevation_angle
    |  compute beam_centre_range
    |  compute DI[i].beam_centre.lat
    |  compute DI[i].beam_centre.lon
    ELSEIF location.loc_method for current_dwell
    = c_spd_boresight_offset_method THEN
    |  compute current_rra_rot_angle
    |  compute beam_centre_elevation_angle
    |  IF beam centre is not on earth THEN
    |  |  use f_sutl_elog() to log "Invalid dwell geometry, dwell #[i]"
    |  |  trap to i_sbit_cleanup()
    |  ENDIF
    |  compute beam_centre_azimuth_angle
    |  slant_range = sgen_ele_ang_range(beam_centre_elevation_angle)
    |  compute DI[i].beam_centre.lat
    |  compute DI[i].beam_centre.lon
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*======================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_cell_valid.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_cell_valid     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------  ------   ------------------------------------
n/a   08-APR-1993  mdb     Initial Release

---------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_cell_valid

PURPOSE:        This function determines whether an individual precipitation
cell has an affect on the attenuation during the current
CPI.

INPUTS:         precipitation cell information

OUTPUT:         c_spd_true, if cell should be modelled
c_spd_false, if cell does not need to be modelled

EXCEPTIONS:     Whenever a fatal error is encountered,  i_sbit_cleanup()
is executed, and further processing is abandoned.

======================================================================*/


void f_sbit_cell_valid ()
    /* t_spd_boolean f_sbit_cell_valid (t_cend_precip_cell cell)   */
    /*% f_sbit_cell_valid function  */

{

/*|

    IF cell affects current burst THEN
<-------return (c_spd_true)
    ELSE return (c_spd_false)
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_cgep_store.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_cgep_store     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author  Change made
---   ----------  ------  ----------------------------------
n/a   08-APR-1993  mdb    Initial Release

---------------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_cgep_store

PURPOSE:        This function stores a block of data to the RSS Processing
parameters file.  The RSS Processing parameters file is
maintained in XDR format in an external UNIX file.  Information
is built on a CPI-by-CPI basis and stored on a
dwell-by-dwell basis.

INPUT:          from global data

OUTPUT:         XDR-encoded records to an XDR format external UNIX file.

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.
===========================================================================*/


void f_sbit_cgep_store ()
    /*% f_sbit_cgep_store function */

{

/*|

    update control block information (t_cgep_control_block)
    write data (t_cgep_DI_elm) to XDR file datablock


*/
```

E-156

```

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_ionosphere.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_ionosphere     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   --------------------------------------
n/a   08-APR-1993  mdb     Initial Release

---------------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_ionosphere

PURPOSE:        This function is called during each burst initialization
to build the ionospheric model coefficients.

INPUTS:         from global data
t_cend_env_def
t_sgen_run_constants
t_sgen_run_parameters

OUTPUTS:        to global data: t_sgen_ionosphere_model_coeffs

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_sbit_ionosphere ()
   /*% f_sbit_ionosphere function  */

{

/*|

   compute geomag_field
   in t_sgen_ionosphere_model_coeffs,
   compute t_sgen_ionosphere_model_coeffs.faraday_rot
   compute t_sgen_ionosphere_model_coeffs.ion_atten
   compute seq_length, dt_ion
```

```
    generate sparse_ionos, the sparse ionospheric sequence

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_jammer_db.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_jammer_db      1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   -----------------------------------------
n/a   08-APR-1993  mdb     Initial Release

--------------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_jammer_db

PURPOSE:        This function is called once per CPI to build the jammer
database.

INPUT:          from global data
t_sgen_run_parameters
t_sgen_run_constants
t_cend_env_def

OUTPUT:         to global data:
t_sgen_jammer_database

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

=========================================================================*/

void f_sbit_jammer_db ()
   /*% f_sbit_jammer_db function  */

{

/*|

   FOR jammers 1 to t_cend_env_def.jammer_count
   LOOP
   |  jammer_validity = f_sbit_target_valid()
   |  NOTE: not a typo; the same function is used as for targets
```

```
    |  IF jammer_validity = c_spd_true THEN
    |  |  Compute jammer_range, jammer_attenuation
    |  ENDIF
    ENDLOOP by jammer

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

877

E-16.2

```
/*=====================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_noise_db.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_noise_db     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ------------------------------
n/a   08-APR-1993  mdb     Initial Release

-----------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_noise_db

PURPOSE:        This function is called once per CPI to build the noise
tables and variables, as well as to generate the noise-related
elements of the DSP Processing Parameters.

INPUT:          from global data

OUTPUT:         to global data: t_sgen_noise_coeffs

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

=====================================================================*/

void f_sbit_noise_db ()
   /*% f_sbit_noise_db function  */

{

/*|

   Compute t_sgen_noise_coeffs.noise_covar matrix between antennas
   t_cgep_DI_elm.noise_power_main = noise_covar[0][0]
   t_cgep_DI_elm.noise_power_aux =
   noise_covar[number_of_antennas-1][number_of_antennas-1]
   Compute t_sgen_noise_coeffs.ucbol
   Compute t_sgen_cpi_database.delta_rf, which is the covariance for
   the current CPI minus the covariance for the previous CPI
```

```
    (factor by factor difference)
    update t_sgen_noise_coeffs.prev_rf

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_rd_map.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_rd_map 1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------  ------   ------------------------------------
n/a   08-APR-1993   mdb    Initial Release

--------------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_rd_map

PURPOSE:        This function is called once per CPI, and builds the
range-doppler map (a transitory product) and the earth
clutter database for the CPI.

This function is compute-intensive, and makes use of the CSPI
coprocessor to vectorize calculations across the largest
possible subsets of the address space.

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_sbit_rd_map ()
    /*% f_sbit_rd_map function  */

{

/*|

    compute t_sgen_cpi_database.max_dop_angle
    compute t_sgen_cpi_database.num_isodops
    compute t_sgen_num_iso_elements
    compute t_sgen_cpi_database.num_isodops

    ##Host side##

```

```
Setup the data pointers for the first range swath
call cvmov() to transfer the data to the CSPI
call CSPI section to process the data

FOR each pair of range swaths (2 onward)
LOOP
|  setup the data pointers for the first swath in the pair
|  call cvmov() to transfer the data to the CSPI
|  call CSPI section to process the data
|  call cvmov() to transfer the data of the previous swath to the Host
|  setup the data pointers for the second swath in the pair
|  call cvmov() to transfer the data to the CSPI
|  call CSPI section to process the data
|  call cvmov() to transfer the data of the previous swath to the Host
ENDLOOP for range swath
call cvmov() to transfer data of the final swath to the CSPI
call CSPI section to process the data
call cvmov() to transfer the data back to the Host
call xldone() for synchronization

##CSPI side##

Compute number-of-patches
Compute number-of-type-5-patches, and store in t_sgen_ec_database.cpc[i][4]
Compute the midswath-elevation-angle
Compute the midswath-grazing-angle and store in t_sgen_ec_database.graz[i]
Compute the midswath range and store in t_sgen_ec_database.range[i]
Compute midswath-range-squared

FOR each type-5 patch (vectorized)
LOOP
```

```
   |  Compute area of type 5 patch and store in t_sgen_ec_database.cpa element
   ENDLOOP

   FOR each non-type-5 patch (vectorized)
   LOOP
   |  Determine patch type and parameters required for area calculation
   |  Update the number of type j patches in t_sgen_ec_database.cpc[i][j]
   |  Compute area of patch and store in t_sgen_ec_database.cpa element
   ENDLOOP

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_target_db.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_target_db      1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------  ------   -----------------------------------
n/a   08-APR-1993  mdb     Initial Release

---------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_target_db

PURPOSE:        This function is called once per CPI to build the target
database for the current CPI.

INPUT:          from global data

OUTPUT:         to global data:
t_sgen_target_db

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_sbit_target_db ()
    /*% f_sbit_target_db function  */

{

/*|

    FOR targets 1 to t_cend_env_def.target_count
    LOOP
    |  target_validity = f_sbit_target_valid()
    |  IF target_validity = c_spd_true THEN
    |  |  Compute target_range, target_attenuation
    |  ENDIF
    ENDLOOP by target

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_chaff_db.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_chaff_db       1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR    Date        Author    Change made
---  -----------  ------  ----------------------------------------
n/a  08-APR-1993   mdb     Initial Release

--------------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_chaff_db

PURPOSE:        This function is called once per CPI, to build the
chaff clutter database for the current CPI.

INPUT:          from global data

OUTPUT:         to global data: t_sgen_chaff_db

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_sbit_chaff_db ()
    /*% f_sbit_chaff_db function  */

{

/*|

    FOR chaff bundles 1 to t_cend_env_def.chaf_count
    LOOP
    |  IF the chaff bundle affects the current frequency THEN
    |  |  increment t_sgen_chaff_db.nchaffs, the number of chaff bundles
    |  |  affecting the current CPI
    |  |  in t_sgen_chaff_db, compute chaf_lo, fcrollof, fcpeak
    |  |  IF cell is inside the atmosphere THEN
    |  |  |  compute t_sgen_chaff_db.chaf_atmos_atten
```

```
  | | ELSE
  | | | t_sgen_chaff_db.chaf_atmos_atten = 1
  | | ENDIF
  | | Compute chaff_rcs
  | ENDIF
  ENDLOOP by chaff bundles

*/

  /* Local macros */

  /* Local constants */

  /* Local Structure Definitions */

  /* Local Variables */

  /* Output Messages */

  /* Code */

}
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_precip_db.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_precip_db      1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ----------------------------------------
n/a   08-APR-1993  mdb     Initial Release

--------------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_precip_db

PURPOSE:        This function is called once per CPI, to build the
precipitation clutter database for the current CPI.

The function is compute-intensive, and makes use of the CSPI
coprocessor.

INPUT:          from global data

OUTPUT:         to global data: t_sgen_precip_db

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

=========================================================================*/

void f_sbit_precip_db ()
    /*% f_sbit_precip_db function  */

{

/*|

    ##Host side##

    Setup the data pointers for the first precipitation cell
    call cvmov() to transfer the data of the first cell to the CSPI
    call CSPI section to process the data
```

```

FOR each pair of precipitation cells (2 to t_cend_env_def.cell_count-2, step 2)
LOOP
|  setup the data pointers for the first cell in the pair
|  call cvmov() to transfer the data to the CSPI
|  call CSPI section to process the data
|  call cvmov() to transfer the data of the previous cell to the Host
|  setup the data pointers for the second cell in the pair
|  call cvmov() to transfer the data to the CSPI
|  call CSPI section to process the data
|  call cvmov() to transfer the data of the previous cell to the Host
ENDLOOP for precipitation cell
call cvmov() to transfer data of the final cell to the CSPI
call CSPI section to process the data
call cvmov() to transfer the data back to the Host
call xldone() for synchronization

##CSPI side##

IF (f_sbit_precip_valid() = c_spd_true) THEN
|  Compute the cell geometry, cell representation and bounding equi-ranges
|  Compute reflectivity and path length through cell
|  FOR each intercepted range swath (vectorized)
|  LOOP
|  |  derive the swath description
|  |  FOR each quadrant representation
|  |  LOOP
|  |  |  IF the cell extends into or is within the quadrant THEN
|  |  |  |  derive the isodop extents
|  |  |  |  FOR each doppler swath (vectorized)
|  |  |  |  LOOP
```

```
    | | | | | Determine the number of scatterers in the isodop and
    | | | | | the patch type
    | | | | | compute t_sgen_precip_db.pscatloc, .precip_rcs
    | | | | ENDLOOP for doppler swath
    | | | ENDIF
    | | ENDLOOP for quadrant representation
    | ENDLOOP for intercepted range swath
    | Compute t_sgen_precip_db.frolloff, .fpeak, .atmos_atten
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:     sbit_precip_valid.c

Identification: [@(#)]

SCCS Id:       @(#)sbit_precip_valid   1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:      Marg Baskin

Change History:

SCR   Date        Author   Change made
--- ----------- ------ ------------------------------------
n/a  08-APR-1993  mdb     Initial Release

----------------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_precip_valid

PURPOSE:        This function "validates" a precipitation cell, i.e., it
determines whether or not the cell needs to be modelled
during the current burst.

INPUT:          from global data

OUTPUT:         TRUE, if the cell must be modelled
FALSE otherwise

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_sbit_precip_valid ()
    /* t_spd_boolean f_sbit_precip_valid ()   */
    /*% f_sbit_precip_valid function  */

{

/*|

    IF cell must be modelled THEN
<------return (c_spd_true)
    ELSE
<-------return (c_spd_false)
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=============================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbit_waveform.c

Identification: [@(#)]

SCCS Id:        @(#)sbit_waveform       1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------  -----  --------------------------------
n/a  08-APR-1993  mdb     Initial Release

-----------------------------------------------------------------

CSU_NAME:       sbit (burst initialization)

FUNCTION:       f_sbit_waveform

PURPOSE:        This function is called once per CPI to generate the waveform
coefficients, waveform related working parameters and
waveform-related elements of the DSP Signal Processing
Parameters for the CPI.

INPUT:          from global data

OUTPUT:         to global data

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbit_cleanup()
is executed, and further processing is abandoned.

=============================================================================*/

void f_sbit_waveform ()
    /*% f_sbit_waveform function  */

{

/*|

    IF starting a new dwell (current_nci = 1) THEN
    |   use f_sbit_cgep_store() to store t_cgep_DI_elm for the previous DI
    ENDIF
    compute t_sgen_cpi_database.radar_wavelength
    compute t_sgen_waveform_coeffs.trans_start_time
    in t_cgep_RSS_process_parms, compute ref_range, look_depression,
look_azimuth, earth_rad_vel, platfm_rad_vel

IF starting a new dwell (current_nci = 1) THEN
|  set t_sgen_waveform_coeffs.adjusted_prf from t_crad_sat_def for the dwell
|  IF t_crad_rxfe.motion_comp = c_spd_true THEN
|  |  recompute adjusted_prf
|  ENDIF
ENDIF

IF starting a new NCI (current_cpi = 1) THEN
|  compute t_sgen_nci_database.coh_burst_time
ENDIF
compute t_sgen_cpi_database.doppler_quant
compute t_sgen_waveform_coeffs.trans_end_time

FOR each DPCA group (1 to t_crad_CFA_antenna.num_DPCA_grps)
LOOP
|  FOR each phase centre in group (1 to t_crad_CFA_antenna.num_ph_ctr_per_grp)
|  LOOP
|  |  compute start_tx_rx_time, end_tx_rx_time
|  |  IF current phase centre is the leading phase centre THEN
|  |  |  in t_cgep_RSS_process_parms, compute CPI_samplg_ofst,
|  |  |  rx_tm_start_burst,
|  |  |  num_samp_input (= t_sgen_waveform_coeffs.num_samples)
|  |  |  in t_sgen_waveform_coeffs, compute jammer_samples, clutter_samples,
|  |  |  receive_start_time_ref, receive_end_time_ref
|  |  |  compute abs_burst_receive_start_time
|  |  ENDIF
|  |  initialize inter_pc_time_delays
|  ENDLOOP by phase centre
|  compute inter_pc_time_delays
```

```
    ENDLOOP by DPCA group

    IF starting a new dwell (current_nci = 1) THEN
    |  compute t_sgen_waveform_coeffs.pulse_compression_replica
    |  compute t_cgep_DI_elm.pulse_replica, the decimated replica for storage
    |  initialize rx_tm_start_burst
    |  set t_sgen_nci_database.receive_ref_time to abs_burst_receive_start_time
    ENDIF

    IF starting a new NCI (current_cpi = 1) THEN
    |  compute t_cgep_NCI_elm.inter_ph_ctr_sep
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
                                         TBD CROSS REFERENCE REPORT
=====================================================================================================
=============================
PAGE NO.   LINE NO.          ENCLOSING FUNCTION

  NO TBDs TO REFERENCE
```

E6 Burst Modelling (SBMT) CSU

THIS PAGE INTENTIONALLY LEFT BLANK

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□                    □□□□
□□□□                    □□□□
□□□□                    □□□□
 □□                      □□
 □□        CDADL PROCESSOR        □□
 □□                      □□
 □□                      □□
 □□         VERSION 2.4.6          □□
 □□                      □□
 □□      SERIAL NUMBER: C38U0011        □□
 □□                      □□
 □□        AUTHORIZED USER:        □□
 □□        Raytheon - Canada        □□
 □□                      □□
 □□                      □□
 □□                      □□
 □□                      □□
 □□                      □□
□□□□                    □□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

1 FILE sbmt.c
3 FILE sbmt_cleanup.c
4 FILE sbmt_ad.c
6 FILE sbmt_discretes.c
8 FILE sbmt_eclutter.c
10 FILE sbmt_if_filter.c
12 FILE sbmt_interpolate.c
14 FILE sbmt_iq.c
16 FILE sbmt_jammers.c
19 FILE sbmt_noise.c
21 FILE sbmt_non_linear.c
23 FILE sbmt_sparse_orbit.c
24 FILE sbmt_store.c
25 FILE sbmt_store_int.c
26 FILE sbmt_targets.c
28 FILE sbmt_precip.c

30 FILE sbmt_chaff.c
32 REPORT Requirements Traceability
33 REPORT INVOCATION TREE
34 REPORT ERROR SUMMARY

```
/*===========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:     sbmt.c

Identification: [@(#)]

SCCS Id:       @(#)sbmt       1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:      Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ---------   -----   ---------------------------------
n/a  08-APR-1993   mdb     Initial Release

-----------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       sbmt_main

PURPOSE:        This function localizes all the processing necessary to
model one complete burst worth of data. It produces as
output one CPI worth of Simulated radar signal data,
and selected intermediate products data which are stored to
disk.

This task is activated by a fork()/execcv() within sgen.
It takes all its parameters from shared global data, and
uses shared global data to return its processing status
to sgen.

EXCEPTIONS:   Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

===========================================================================*/

#include "sbmt_priv.h"

void sbmt_main ()
    /*% sbmt_main function */

{

/*|

    use f_sbmt_sparse_orbit() to generate a sparse sequence of orbital positions
    use f_sbmt_eclutter() to generate the earth clutter samples
    use f_sbmt_interpolate() to interpolate the samples to an oversampled matrix
    use f_sbmt_precip() to generate a matrix of precipitation clutter samples and
    add them to the aggregate signal
    use f_sbmt_chaff() to generate a matrix of chaff cell clutter samples and
    add them to the aggregate signal
    use f_sbmt_discretes() to generate a matrix of discrete clutter samples and
    add them to the aggregate signal
    use f_sbmt_targets() to generate a matrix of target samples and
    add them to the aggregate signal
    use f_sbmt_jammers() to generate a matrix of jammer samples and
    add them to the aggregate signal
    use f_sbmt_noise() to generate a matrix of noise clutter samples and
    add them to the aggregate signal
    use f_sbmt_non_linear() to factor in non-linearities to the aggregate signal
    use f_sbmt_if_filter() to factor I/F filter affects to the aggregate signal
    use f_sbmt_iq() to apply I/Q to the aggregate signal
    use f_sbmt_ad() to apply A/D to the aggregate signal
    use f_sbmt_store() to store the aggregate signal

    IF (t_sgen_number_of_cpis = t_sgen_cpi_counter) THEN
    |   close each of the open output product files
    ENDIF

    set t_sgen_active_status = s_sgen_sbmt_finished

*/

    /* Local macros */

    /* Local constants */

```

```
	/* Local Structure Definitions */

	/* Local Variables */

	/* Output Messages */

	/* Code */

}

```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_cleanup.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_cleanup        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------- ------  ----------------------------------------
n/a  08-APR-1993   mdb     Initial Release

---------------------------------------------------------------------------

CSU_NAME:       sbmt (Burst modelling)

FUNCTION:       i_sbmt_cleanup

PURPOSE:        This function performs clean-up activities prior to the
termination of sbmt. This includes deallocation of
memory allocated to the global database, closing and
clean-up of intermediate files, etc.

It is set up as an interrupt handler, and is entered
by sbmt and and any of its subfunctions upon recognition of an
unrecoverable error condition.

EFFECTS:        sets t_sgen_active_status = c_sgen_sbmt_terminated

==========================================================================*/

void i_sbmt_cleanup ()
    /*% i_sbmt_cleanup function */

{

/*|

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:    sbmt_ad.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_ad   1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  -----------  ------  ----------------------------------------
n/a  08-APR-1993   mdb     Initial Release

--------------------------------------------------------------------

CSU_NAME:    sbmt (Burst Modelling)

FUNCTION:    f_sbmt_ad

PURPOSE:    This function is called once per CPI, and applies to A/D
front-end factors to the SRSD samples for the current CPI.

This function uses the CSPI coprocessor to vectorize operations
and perform them in parallel.

EXCEPTIONS:    Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

=========================================================================*/

void f_sbmt_ad ()
    /*% f_sbmt_ad function */

{

/*|

    ## Split Programming Mode -- Host Section ##
    Setup data pointer
    use cvmov () to transfer the data of the first channel from Host to CSPI
    use CSPI section to do calculation on this data
    FOR each pair of channels
    LOOP
    |
    |  ## using Double-Buffering ##
    |  setup data pointer
    |  use cvmov () to transfer the data of the current channel from Host to CSPI
    |  use CSPI section to do calculation on this data
    |  use cvmov () to transfer the data of the last channel from CSPI to Host
    |  setup data pointer
    |  use cvmov () to transfer the data of the next channel from Host to CSPI
    |  use CSPI section to do calculation on this data
    |  use cvmov() to transfer the data of the current channel from CSPI to Host
    ENDLOOP by channel
    setup data pointer
    use cvmov () to transfer the data of the last channel from Host to CSPI
    use CSPI section to do calculation on this data
    use cvmov () to transfer this data from CSPI to Host
    use xldone () for synchronization

    IF capture_CPI_enable = c_spd_true THEN
    |  use f_sbmt_store_int(c_cgep_rss_post_ad) to update t_cgep_control_block
    |  and store the required portions of t_sgen_srsd_samples
    ENDIF

    ## END by Host Section ##

    ## Split Programming Mode – CSPI Section ##

    Factor the A/D filter values into t_sgen_srsd_samples for each channel

    ## END by CSPI Mode ##


*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:    sbmt_discretes.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_discretes   1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:      Marg Baskin

Change History:

SCR    Date        Author    Change made
---  ----------  -----  ---------------------------------
n/a  08-APR-1993   mdb      Initial Release

-------------------------------------------------------------------------

CSU_NAME:    sbmt (Burst Modelling)

FUNCTION:    f_sbmt_discretes

PURPOSE:    This function is called once per CPI.  It generates a
matrix of samples representing the composite signals
from all discrete scatterers, and adds them to the
oversampled composite matrix for the CPI.

This function is compute-intensive, and makes use fo the CSPI
coprocessor to vectorize processing of data elements.

INPUT:          from global data

OUTPUT:         modified t_sgen_aggregate_oversamples

EXCEPTIONS:    Whenever a fatal error is encountered, i_sbmt_cleanup() is
executed, and further processing is abandoned.

=========================================================================*/

void f_sbmt_discretes ()
    /*% f_sbmt_discretes function */

{

/*|
    ## Split Programming Mode -- Host Section ##
    initialize t_sgen_oversamples
    setup data pointer
    use cvmov () to transfer the data of the first discrete from Host to CSPI
use CSPI section to do calculation on this data
FOR each pair of discretes (2 to #ofdiscretes - 2  step 2)
LOOP
|   setup data pointer
|   use cvmov () to transfer the data of the current discrete from Host to CSPI
|   use CSPI section to do calculation on this data
|   use cvmov () to transfer the data of the last discrete from CSPI to Host
|   setup data pointer
|   use cvmov () to transfer the data of the next discrete from Host to CSPI
|   use CSPI section to do calculation on this data
|   use cvmov () to transfer the data of the current discrete from CSPI to Host
ENDLOOP by discretes

setup data pointer
use cvmov () to transfer the data of the last discrete from Host to CSPI
use CSPI section to do calculation on this data
use cvmov () to transfer this data from CSPI to Host
use xldone () for synchronization

FOR each channel
LOOP
|   add t_sgen_oversamples to t_sgen_aggregate_oversamples
ENDLOOP by channel

## END by Host Section ##

## Split Programming Mode -- CSPI Section ##

Compute discrete range, location, grazing angle, azimuth and elevation
Compute total precipitation attenuation
Compute constant loss
```

4

```
    Compute burst receive start time for the discrete
    Generate the sparse sequence for the discrete
    Apply the TxAntennaGain to the sparse sequence
    IF ionosphere is "ON" THEN
    |  Apply ionospheric scintillation effects
    ENDIF
    Apply precipitation attenuation and propagation loss to the
    sparse sequence
    ## Vectorize antenna -- FOR each antenna ##
    Apply receiving antenna gain to the sparse sequence
    ## END by vectorize antenna ##

    ## Vectorize antenna -- FOR each antenna ##
    Compute sparse sequence of burst receive sample time instances
    Compute sparse sequences of discrete sample times and
    transmit instances
    Compute sequence of waveform samples for the transmit times
    Interpolate to generate a full sequence of discrete samples
    Compute the discrete return
    add the discrete return to t_sgen_oversamples
    FOR each antenna
    LOOP
    |  ## Vectorize tap off --  FOR each tap off the current antenna ##
    |  Compute sequence of waveform samples for the transmit times
    |  Interpolate to generate a full sequence of discrete samples
    |  Compute the discrete return
    |  Add the discrete return to t_sgen_oversamples
    |  ## END by tap off ##
    ENDLOOP by antenna
    ## End by vectorize antenna ##

    ## END by CSPI Section ##

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

E-1-15

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_eclutter.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_eclutter     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------  ------  ----------------------------------------
n/a  08-APR-1993   mdb     Initial Release

--------------------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       f_sbmt_eclutter

PURPOSE:        This function is called once per CPI.  It generates a matrix
of earth clutter samples for the CPI and (optionally) stores
these samples as an intermediate product in an external
UNIX file in XDR format.

This function is computation-intensive, and will make use of
the CSPI mathematical coprocessor to vectorize sequences of
computations on the sample space, either performing these
calculations for all samples simultaneously, or in cases where
the sample matrix is too large for the CSPI memory, breaking
the samples matrix down and processing them in segments.

INPUT:          from global data

OUTPUT:         t_sgen_earth_clutter_samples, optionally stored to an external
UNIX file.

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.


==========================================================================*/
void f_sbmt_eclutter ()
    /*% f_sbmt_eclutter function */

{
/*|
allocate (initialized)
t_sgen_earth_clutter_samples[clutter_samples][number_of_channels]

## Split Programming Mode -- Host Section ##
setup data pointer
use cvmov () to transfer data of the first isodop from Host to CSPI
use CSPI section to do calculation on this data
FOR each pair of isodops  (2 to number of isodop - 2 step 2)
LOOP
|  ## Using Double-Buffering ##
|  setup data pointer
|  use cvmov () to transfer the data of the current isodop
|  from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov () to transfer the data of the last isodop
|  from CSPI to HOST
|  setup data pointer
|  use cvmov () to transfer the data of the next isodop
|  from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov () to tranfer data of the current isodop
|  from CSPI to Host
ENDLOOP by isodop

setup data pointer
use cvmov () to transfer the data of the last isodop from Host to CSPI
use CSPI section to do calculation on this data
use cvmov () to transfer this data from CSPI to Host
use xldone () for synchronization

t_cgep_filetype = c_cgep_rss_clut
```

```

update t_cgep_control_block and store a portion of the
generated matrix, based on the hierarchy of capture
flags into corresponding XDR output stream

## END by Host Section ##

## Split Programming Mode -- CSPI Section ##

Compute the number of scatterers in the isodop
Compute the number of range bins in the isodop
FOR each range bin
LOOP
|  Compute the number of scatterers in the current range bin
|  Initialize the antenna sparse sequence for each antenna
|  FOR all scatters or segment of scatterers
|  LOOP
|  |  Compute the absolute scatterer number, its range
|  |  and its grazing angle
|  |  Compute the R-squared propagation loss to the
|  |  scatterer
|  |  Compute its atmospheric attenuation
|  |  Compute the scatterer location, angle and number of symmetries
|  |  ## Vectorize symmetry -- FOR each symmetry ##
|  |  Compute clutter type, peak frequency, spectral width,
|  |  scatterer reflection coefficient, and scatterer distribution
|  |  Compute total precipitation attenuation, total constant loss,
|  |  TxAntennaGain
|  |  Determine burst receive start time for the current scatterer
|  |  Compute RxAntennaGain, receive power
|  |  IF the receive power is greater than the power threshold for
|  |  all antennas THEN
|  |  |  Generate a sparse sequence for the scatterer
|  |  |  Apply the TxAntennaGain to the Sparse sequence
|  |  |  IF ionosphere is "ON" THEN
|  |  |  |  Apply the ionospheric scintillation effects
|  |  |  ENDIF
|  |  |  apply precipitation attenuation and propagation loss
|  |  |  FOR each symmetry
|  |  |  LOOP
|  |  |  |  ## Vectorize antenna -- FOR each antenna ##
|  |  |  |  apply the receiving antena gain to the receive sparse
|  |  |  |  sequence
|  |  |  |  add current sequence to the sparse antenna sequence matrix
|  |  |  |  ## END by vectorize antenna ##
|  |  |  ENDLOOP by symmetry
|  |  ENDIF
|  |  ## END by vectorize symmetry ##
|  ENDLOOP by scatterer grouping
|
|  ## Vectorize antenna -- FOR each antenna ##
|  Determine the sparse sequence of burst receive sample times
|  Determine the sparse sequence of scatterer sample times and
|  transmit instances
|  Determine the full sequence of scatterer samples
|  Determine the scatterer return
|  Add the scatterer return to t_sgen_earth_clutter_samples
|  FOR each antenna
```

```
    | LOOP
    | | ## Vectorize delay tap -- FOR each delay tap ##
    | | Determine the sequence of waveform samples for the transmit
    | | instances
    | | Determine the full sequence of scatterer samples
    | | Determine the scatterer return
    | | Add the delay tap return to t_sgen_earth_clutter_samples
    | | ## END by vectorize delay tap ##
    | ENDLOOP by anttenna
    | ## END by vectorize antenna ##
    ENDLOOP by range bin

    ## END by CSPI section ##
*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_if_filter.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_if_filter      1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date         Author   Change made
---  ----------- ------  ---------------------------------------
n/a  08-APR-1993  mdb     Initial Release

----------------------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       f_sbmt_if_filter

PURPOSE:        This function is called once per CPI.  It applies IF filter
factors to the oversampled composite samples matrix for the
CPI and (optionally) stores the resulting composite matrix
as an intermediate product.

This function uses the CSPI coprocessor to vectorize
operations across the sample space and perform them in
parallel.

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_sbmt_if_filter ()
    /*% f_sbmt_if_filter function */

{

/*|
    ## Split Programming Mode -- Host Section ##
    Setup data pointer
    use cvmov () to transfer the data of the first channel from Host to CSPI
    use CSPI section to do calculation on this data
    FOR each pair of channels
    LOOP
    |
| ## using Double-Buffering ##
| setup data pointer
| use cvmov () to transfer the data of the current channel from Host to CSPI
| use CSPI section to do calculation on this data
| use cvmov () to transfer the data of the last channel from CSPI to Host
|
| setup data pointer
| use cvmov () to transfer the data of the next channel from Host to CSPI
| use CSPI section to do calculation on this data
| use cvmov() to transfer the data of the current channel from CSPI to Host
ENDLOOP by channel
setup data pointer
use cvmov () to transfer the data of the last channel from Host to CSPI
use CSPI section to do calculation on this data
use cvmov () to transfer this data from CSPI to Host
use xldone () for synchronization

IF capture_CPI_enable = c_spd_true THEN
| use f_sbmt_store_int(c_cgep_rss_if_filt) to update t_cgep_control_block
| and store the required portions of t_sgen_aggregate_oversamples
ENDIF

## END by Host Section ##

## Split Programming Mode -- CSPI Section ##

Factor the IF filter values into t_sgen_aggregate_oversamples elements
Generate an output derivative sequence
decimate the oversampled values by selecting a subsegment
store the output segment into t_sgen_srsd_samples
decimate the output derivative and store in a derived matrix
```

```

    ## END by CSPI Section ##

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

E-222

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:     sbmt_interpolate.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_interpolate    1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR    Date          Author    Change made
---    ----------    -----    -----------------------------------
n/a    08-APR-1993   mdb      Initial Release

--------------------------------------------------------------------------

CSU_NAME:        sbmt (Burst Modelling)

FUNCTION:        f_sbmt_interpolate

PURPOSE:         This function is called once per CPI.  It takes the earth
clutter samples matrix and interpolates it to an oversampled
matrix.

This function uses the CSPI coprocessor to perform compute
intensive internal calculations associated with producing
the interpolated oversamples.

INPUTS:          from global data

OUTPUTS:         t_sgen_aggregate_oversamples, an oversampled matrix for the CPI

EXCEPTIONS:      Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

=========================================================================*/

void f_sbmt_interpolate ()
    /*% f_sbmt_interpolate function */

{

/*|

    oversamples = MAX (jammer_noise_factor, clutter_factor)
    allocate
    t_sgen_aggregate_oversamples[num_samples*oversamples][number_of_channels]
```

```

## Split Programming Mode -- Host Section ##
Setup data pointer
use cvmov () to transfer the data of the first channel from Host to CSPI
use CSPI section to do calculation on this data
FOR each pair of channels
LOOP
|  ## using Double-Buffering ##
|  setup data pointer
|  use cvmov () to transfer the data of the current channel from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov () to transfer the data of the last channel from CSPI to Host
|
|  setup data pointer
|  use cvmov () to transfer the data of the next channel from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov () to transfer the data of the current channel from CSPI to Host
ENDLOOP by channel
setup data pointer
use cvmov () to transfer the data of the last channel from Host to CSPI
use CSPI section to do calculation on this data
use cvmov () to transfer this data from CSPI to Host
use xldone () for synchronization

## END by Host Section ##

## Split Programming Mode -- CSPI Section ##

divide t_sgen_earth_clutter_samples into manageable segments, based
on the processing limits of the CSPI
## Vectorize segment -- FOR each segment ##
```

```
    Generate the interpolated samples and store in t_sgen_aggregate_oversamples
    ## END by vectorize segment ##

    ## END by CSPI Section ##

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_iq.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_iq     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ---------------------------------------
n/a   08-APR-1993  mdb     Initial Release

---------------------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       f_sbmt_iq

PURPOSE:        This function is called once per CPI.  It takes the composite
samples matrix after decimation and applies I/Q distortions
to its elements.  Optionally, it stores the resulting
matrix as an intermediate product.

This function uses the CSPI coprocessor to vectorize operations
across the sample space.

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

=========================================================================*/

void f_sbmt_iq ()
    /*% f_sbmt_iq function */

{

/*|

    ## Split Programming Mode -- Host Section ##
    Setup data pointer
    use cvmov () to transfer the data of the first channel from Host to CSPI
    use CSPI section to do calculation on this data

    FOR each pair of channels
    LOOP
```

```
|
|  ## using Double-Buffering ##
|  setup data pointer
|  use cvmov () to transfer the data of the current channel from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov () to transfer the data of the last channel from CSPI to Host
|
|  setup data pointer
|  use cvmov () to transfer the data of the next channel from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov() to transfer the data of the current channel from CSPI to Host
ENDLOOP by channel

setup data pointer
use cvmov () to transfer the data of the last channel from Host to CSPI
use CSPI section to do calculation on this data
use cvmov () to transfer this data from CSPI to Host
use xldone () for synchronization

IF capture_CPI_enable = c_spd_true THEN
|  use f_sbmt_store_int(c_cgep_rss_post_sd) to update t_cgep_control_block
|  and store the required portions of t_sgen_srsd_samples
ENDIF


## END by Host Section ##

## Split Programming Mode -- CSPI Section ##

Factor the I/Q distortions into t_sgen_srsd_samples

```

```
    ## END by CSPI Section ##


*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}

```

E-508

```
/*===========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_jammers.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_jammers        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------- ------  ------------------------------------
n/a  08-APR-1993  mdb    Initial Release

----------------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       f_sbmt_jammers

PURPOSE:        This function is called once per CPI.  It generates an
oversampled matrix of signals representing the effects of
each jammer within the run and (optionally) stores one or
more of them as intermediate products.  It adds each jammer's
effects to the oversampled composite matrix for the CPI.

This function is compute-intensive and makes use of the CSPI
coprocessor to vectorize operations and perform them across
the sample space in parallel.

EXCEPTIONS:   Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

===========================================================================*/

void f_sbmt_jammers ()
    /*% f_sbmt_jammers function */

{

/*|
    ## Split Programming Mode -- Host Section ##
    Initialize t_sgen_oversamples
    Setup data pointer
    use cvmov () to transfer the data of the first jammer from Host to CSPI
    use CSPI section to do calculation on this data
    FOR each pair of jammers (2 to t_cend_env_def.jammer_count - 2 step 2)
```

```
LOOP
|  ## using Double-Buffering ##
|  setup data pointer
|  use cvmov () to transfer the data of the current jammer
|  from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov () to transfer the data of the last jammer from CSPI to Host
|
|  setup data pointer
|  use cvmov () to transfer the data of the next jammer from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov() to transfer the data of the current jammer from CSPI to Host
ENDLOOP by jammer

setup data pointer
use cvmov () to transfer the data of the last target from Host to CSPI
use CSPI section to do calculation on this data
use cvmov () to transfer this data from CSPI to Host
use xldone () for synchronization

IF capture_CPI_enable = c_spd_true THEN
|  use f_sbmt_store_int(c_cgep_rss_jam) to update t_cgep_control_block
|  and store the required portions of t_sgen_oversamples
ENDIF

FOR each channel
LOOP
|  Add t_sgen_oversamples to t_sgen_aggregate_oversamples
ENDLOOP
## End by Host section ##
## Split Programming Mode -- CSPI Section ##
## Barrage, Swept, Follower jammers ##

IF the jammer is of type "barrage" or "swept" or "follower" THEN
| IF the jammer is active during the current CPI THEN
| | Compute the start/stop time of period during which the
| | jammer is active
| ENDIF
| divide the active interval down into manageable time segments
| FOR each segment
| LOOP
| | FOR each antenna
| | LOOP
| | | Compute the antenna specific jammer effects modifiers
| | | ## Vectorize channel -- FOR each channel of the current antenna ##
| | | Compute the channel-specific jammer effects modifiers
| | | ## Vectorize tap delay -- FOR each tap delay ##
| | | Use the antenna- and channel-specific modifiers to
| | | generate the jammer data for the current segment,
| | | and store it in t_sgen_oversamples
| | | ## END by vectorize tap delay ##
| | | ## END by vectorize channel ##
| | ENDLOOP by antenna
| ENDLOOP by segment
|
| IF the jammer is of type "swept" THEN
| | ## Vectorize sample-- FOR each sample in the active time segment ##
| | Compute SweepMod, the sweep modifier
| | ## Vectorize channel -- FOR each channel ##
| | Multiply the corresponding t_sgen_oversamples
| | element by SweepMod
| | ## END by vectorize channel ##
| | ## END by vectorize sample ##
| ENDIF
|
| IF the jammer is of type "follower" THEN
| | Compute the jammer's switching time
| | IF the follower jammer's frequency from the previous burst
| | still affects the signal THEN
| | | ## Vectorize time period--FOR the time period from when the jammer
| | | becomes active until it switches frequency ##
| | | Compute the follower modifier
| | | ## Vectorize channel -- FOR each channel ##
| | | Multiply the corresponding t_sgen_oversamples
| | | element by the follower modifier
| | | ## END by vectorize channel ##
| | | ## END by vectorize time period ##
| | ENDIF
| ENDIF
|
| IF the duty cycle of the jammer is less than 100 percent THEN
| | Compute those portions of the CPI during which the jammer is OFF
| | ## Vectorize sample--FOR each t_sgen_oversamples element for
| | which the jammer is OFF ##
| | set the sample to zero
| | ## END by vectorize sample ##
| ENDIF
ENDIF
```

```

    ## Deception Jammers ##

    IF the jammer is of type "deception" THEN
    |  IF the RSS is to generate false targets THEN
    |  |  ## Vectorize false target--FOR the defined number of false targets ##
    |  |  Compute a range offset and velocity offset for the target
    |  |  ## END by vectorize false target ##
    |  ENDIF
    |  ## Vectorize false target -- FOR each false target ##
    |  ## Vectorize channel -- FOR each channel ##
    |  Factor the false target information into t_sgen_oversamples
    |  ## END by vectorize channel ##
    |  ## END by vectorize false target ##
    ENDIF

    ## END by CSPI Section ##
*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

```

```
1084 }
```

E.5.13

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_noise.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_noise  1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date          Author   Change made
---   -----------   ------   --------------------------------------
n/a   08-APR-1993   mdb      Initial Release

--------------------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       f_sbmt_noise

PURPOSE:        This function is called once per CPI.  It generates an
oversampled matrix representing noise (internal and external)
signals.  This matrix is optionally stored as an intermediate
product, then added to the oversampled composite matrix for
the CPI.

This function is compute intensive, and makes use of the CSPI
coprocessor to vectorize and parallel process the sample space.

INPUTS:         from global data

OUTPUTS:        modified t_sgen_aggregate_oversamples
optional UNIX external file

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_sbmt_noise ()
    /*% f_sbmt_noise function */

{

/*|

    ## Split Programming Mode -- Host Section ##
```

```

initalize t_sgen_oversamples
Setup data pointer
use cvmov () to transfer the data of the first antenna from Host to CSPI
use CSPI section to do calculation on this data

FOR a pair of antenna
LOOP
|  ## using Double-Buffering ##
|  setup data pointer
|  use cvmov () to transfer the data of the current antenna from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov () to transfer the data of the last antenna from CSPI to Host
|
|  setup data pointer
|  use cvmov () to transfer the data of the next antenna from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov() to transfer the data of the current antenna from CSPI to Host
ENDLOOP by target
setup data pointer
use cvmov () to transfer the data of the last antenna from Host to CSPI
use CSPI section to do calculation on this data
use cvmov () to transfer this data from CSPI to Host
use xldone () for synchronization
IF capture_CPI_enable = c_spd_true THEN
|  use f_sbmt_store_int(c_cgep_rss_exn) to update t_cgep_control_block
|  and store the required portions of t_sgen_oversamples
ENDIF

FOR each channel
LOOP
```

```
    |  Add t_sgen_oversamples to t_sgen_aggregate_oversamples
    ENDLOOP

    ## END by Host Section ##

    ## Split Programming Mode -- CSPI Section ##

    get the current antenna
    Divide the samples into manageable segments
    FOR each segment
    LOOP
    |  Generate base noise values
    |  ## Vectorize antenna--FOR each antenna ##
    |  ## Use gather/scatter method for IF statement ##
    |  IF there is a Cholesky factor effect between the two
    |  antennas THEN
    |  |  ## Vectorize tap delay -- FOR each tap delay of antenna ##
    |  |  Factor Cholesky factor of noise covariance
    |  |  into noise values
    |  |  ## END by vectorize tap delay ##
    |  ENDIF
    |  ## END by vectorize antenna ##
    ENDLOOP by segment

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_non_linear.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_non_linear     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  ----------  -----  ---------------------------------
n/a  08-APR-1993   mdb     Initial Release

---------------------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       f_sbmt_non_linear

PURPOSE:        This function is called once per CPI.  It applies non-linearity
factors to the oversampled composite signal matrix and
optionally stores the resulting matrix as an intermediate
product.

This function makes use of the CSPI coprocessor to vectorize
calculations and perform them in parallel.

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_sbmt_non_linear ()
    /*% f_sbmt_non_linear function */

{

/*|
    ## Split Programming Mode -- Host Section ##
    Setup data pointer
    use cvmov () to transfer the data of the first channel from Host to CSPI
    use CSPI section to do calculation on this data
    FOR each pair of channel
    LOOP
    |   ## using Double-Buffering ##
    |   setup data pointer
|  use cvmov () to transfer the data of the current channel from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov () to transfer the data of the last channel from CSPI to Host
|
|  setup data pointer
|  use cvmov () to transfer the data of the next channel from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov() to transfer the data of the current channel from CSPI to Host
ENDLOOP by target

setup data pointer
use cvmov () to transfer the data of the last channel from Host to CSPI
use CSPI section to do calculation on this data
use cvmov () to transfer this data from CSPI to Host
use xldone () for synchronization

IF capture_CPI_enable = c_spd_true THEN
|  use f_sbmt_store_int(c_cgep_rss_post_nl) to update t_cgep_control_block
|  and store the required portions of t_sgen_oversamples
ENDIF

## END by Host section ##

## Split Programming Mode -- CSPI Section ##

Apply the non-linearity function to AggregateOversamples for each channel

## END by CSPI Section ##

*/

```

```
    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_sparse_orbit.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_sparse_orbit   1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  -----------  ------  ----------------------------------------
n/a  08-APR-1993   mdb     Initial Release

---------------------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       f_sbmt_sparse_orbit

PURPOSE:        This function is called once per CPI.  It generates a sparse
sequence matrix of SBR orbital positions over the course of
the CPI.  This matrix is interpolated to produce a full
time sequence of orbital positions.

INPUT:          from global data

OUTPUT:         to global data:
t_sgen_sbr_orbit_position vector

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.



==========================================================================*/

void f_sbmt_sparse_orbit ()
    /*% f_sbmt_sparse_orbit function */

{

/*|

    compute delta-t, the sample spacing and N, the number of samples

    generate t_sgen_sbr_orbit_position[1...N], the sparse sequence matrix of SBR
    positions during the current CPI

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*===========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_store.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_store  1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ------------------------------------
n/a   08-APR-1993  mdb     Initial Release

---------------------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       f_sbmt_store

PURPOSE:        This function stores the generated composite samples matrix
for a single CPI into the SRSD product file in the format
and sample ordering required by the Signal Playback activity.

This file is a UNIX external file, not stored in XDR format.

EXCEPTIONS:   Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

===========================================================================*/

void f_sbmt_store ()
    /*% f_sbmt_store function */

{

/*|


*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

```

E-208

```
    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

E-[illegible]

```
/*======================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_store_int.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_store_int      1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ----------------------------------------
n/a   08-APR-1993  mdb     Initial Release

------------------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       f_sbmt_store_int

PURPOSE:        This function decimates and stores required portions of a
generated CPI sample matrix into an XDR format, UNIX external
file

INPUT:          file_id: identifier for which intermediate product file
to store
from global data: number_of_samples, adjusted_prf

OUTPUT:         A block of XDR-encoded output file data.
The file is stored in directory <output_dir>, with a filename
selected on the basis of file_id

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

======================================================================*/

void f_sbmt_store_int ()

    /* void f_sbmt_store_int (t_cgep_filetype file_id) */
    /*% f_sbmt_store_int function */

{

/*|

    FOR each channel (1 to number_of_channels)
    LOOP
    |  IF the capture flags associated with that channel = c_spd_true THEN
    |  |  setup decimation parameters
    |  |  store the selected portion of the samples vector for the channel
    |  ENDIF
    ENDLOOP by channel
    update t_cgep_control_block

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}

```

```
/*============================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_targets.c

Identification: [@(#)]

SCCS Id:      .  @(#)sbmt_targets        1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR    Date        Author   Change made
---  -----------  ------  ------------------------------------
n/a  08-APR-1993   mdb     Initial Release

-----------------------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       f_sbmt_targets

PURPOSE:        This function is called once per CPI.  It generates an
oversampled matrix of signals representing the effects of
each target within the run and (optionally) stores one or
more of them as intermediate products.  It adds each target's
effects to the oversampled composite matrix for the CPI.

This function is compute-intensive and makes use of the CSPI
coprocessor to vectorize samples and process them in parallel.

EXCEPTIONS:   Whenever a fatal error is encountered, i_sbmt_cleanup() is
executed, and further processing is abandoned.

============================================================================*/
void f_sbmt_targets ()
    /*% f_sbmt_targets function */
{
/*|
    ## Split Programming Mode -- Host Section ##
    Initialize t_sgen_oversamples
    Setup data pointer
    use cvmov () to transfer the data of the first target from Host to CSPI
    use CSPI section to do calculation on this data
    FOR each pair of targets (2 to t_cend_env_def.target_count-2 step 2)
    LOOP
    |  ## using Double-Buffering ##
    |  setup data pointer
    |  use cvmov () to transfer the data of the current target
|  from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov () to transfer the data of the last target from CSPI to Host
|
|  setup data pointer
|  use cvmov () to transfer the data of the next target from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov() to transfer the data of the current target from CSPI to Host
ENDLOOP by target
setup data pointer
use cvmov () to transfer the data of the last target from Host to CSPI
use CSPI section to do calculation on this data
use cvmov () to transfer this data from CSPI to Host
use xldone () for synchronization
IF capture_CPI_enable = c_spd_true THEN
|  use f_sbmt_store_int(c_cgep_rss_tar) to update t_cgep_control_block
|  and store the required portion of t_sgen_oversamples
ENDIF

FOR each channel
LOOP
|  Add t_sgen_oversamples to t_sgen_aggregate_oversamples
ENDLOOP

## END by Host section ##

## Split Programming Mode -- CSPI Section ##

Compute target range, aspect, grazing angle, azimuth and elevation
Compute target reflection coefficient and distribution
Compute total precipitation attenuation
```

```
    Compute constant loss
    Compute burst receive start time for the target
    Generate the sparse sequence for the target
    Apply the TxAntennaGain to the space sequence
    IF ionosphere is "ON" THEN
    |  Apply ionospheric scintillation effects
    ENDIF
    Apply precipitation attenuation and propagation loss to the
    sparse sequence
    ## Vectorize antenna -- FOR each antenna ##
    Apply receiving antenna gain to the sparse sequence
    ## END by vectorize antenna ##

    ## Vectorize antenna -- FOR each antenna ##
    Compute sparse sequence of burst receive sample time instances
    Compute sparse sequences of target sample times and transmit
    instances
    Compute sequence of waveform samples for the transmit times
    Interpolate to generate a full sequence of target samples
    Compute the target return
    Add the target return to t_sgen_oversamples
    FOR each antenna
    LOOP
    |  ## Vectorize tap off -- FOR each tap off ##
    |  Compute sequence of waveform samples for the transmit times
    |  Interpolate to generate a full sequence of target samples
    |  Compute the target return
    |  Add the target return to t_sgen_oversamples
    |  ## END by vectorize tap off ##
    ENDLOOP by antenna
    ## END by vectorize antenna ##

    ## END by CSPI section ##
*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=============================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_precip.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_precip 1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ------------------------------------
n/a   08-APR-1993  mdb     Initial Release

-----------------------------------------------------------------------------

CSU_NAME:       sbmt (Burst Modelling)

FUNCTION:       f_sbmt_precip

PURPOSE:        This function is called once per CPI to generate an oversampled
matrix of precipitation clutter signal.

This function makes use of the CSPI mathematical coprocessor
to vectorize data and process it in parallel.

INPUT:          from global data

OUTPUT:         t_sgen_aggregate_oversamples
CPI sample data, stored to an external UNIX file

EXCEPTIONS:     Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

=============================================================================*/

void f_sbmt_precip ()
    /*% f_sbmt_precip function */

{

/*|
    ## Split Programming Mode -- Host Section ##
    allocate and initialize t_sgen_oversamples
    setup data pointer
    use cvmov () to transfer the data of the first precipitation cell
    from Host to CSPI
```

```
use CSPI section to do calculation on this data
FOR each pair of precipitation cells
|  (2 to t_cend_env_def.cell_count - 2 step 2)
LOOP
|  ## Using Double-Buffering ##
|  use cvmov () to transfer the data of the current cell from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov () to transfer the data of the last cell from CSPI to Host
|
|  use cvmov () to transfer the data of the next cell from Host to CSPI
|  use CSPI section to do calculation on this data
|  use cvmov () to transfer the data of the current cell from CSPI to Host
ENDLOOP by precipitation cell

use cvmov () to transfer the data of the last cell from Host to CSPI
use CSPI section to do calculation on this data
use cvmov () to transfer this data from CSPI to Host
use xldone () for synchronization

FOR each channel
LOOP
|  add t_sgen_oversamples to t_sgen_aggregate_oversamples
ENDLOOP by channel

## END by Host Section ##

## Split Programming Mode -- CSPI Section ##

FOR the scatterers within this cell
LOOP
|  Compute the point to point range to the scatterer
```

```
    | Compute total constant loss
    | Compute burst receive start time
    | Generate a sparse scatterer sequence
    | Apply the TxAntennaGain to the sparse sequence
    | IF ionosphere is "ON" THEN
    | | apply ionospheric scintillation to the sparse sequence
    | ENDIF
    | Apply precipitation attenuation and propagation loss to the
    | sparse sequence
    | ## Vectorize antenna -- FOR each antenna ##
    | Apply RxAntennaGain to the sparse sequence
    | ## END by vectorize antenna ##
    | ## Vectorize antenna -- FOR each antenna ##
    | Determine the sparse sequence of burst receive sample time
    | Determine the sparse sequence of scatterer sample times
    | and transmit instances
    | Determine the sequence of waveform samples for the
    | transmit instances
    | Determine the full sequence of scatterer samples
    | Determine the scatterer return
    | Add the scatterer return to t_sgen_oversamples
    | FOR each antenna
    | LOOP
    | | ## Vectorize delay tap -- FOR each delay tap ##
    | | Determine the sequence of waveform samples for the
    | | transmit instances
    | | Determine the full sequence of scatterer samples
    | | Determine the scatterer return
    | | Add the delay tap return to t_sgen_oversamples
    | | ## END by vectorize delay tap ##
    | ENDLOOP by antenna
    | ## END by vectorize antenna ##
    ENDLOOP by the scatterers within this cell

    ## END by CSPI section ##

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*==========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sbmt_chaff.c

Identification: [@(#)]

SCCS Id:        @(#)sbmt_chaff  1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---  -----------  ------  ---------------------------------------
n/a  08-APR-1993   mdb     Initial Release

--------------------------------------------------------------------------

CSU_NAME:        sbmt (Burst Modelling)

FUNCTION:        f_sbmt_chaff

PURPOSE:         This function is called once per CPI to generate an oversampled
matrix of chaff clutter signal.

This function makes use of the CSPI mathematical coprocessor
to vectorize data and process it in parallel.

INPUT:           from global data

OUTPUT:          t_sgen_aggregate_oversamples
CPI sample data, stored to an external UNIX file

EXCEPTIONS:    Whenever a fatal error is encountered, i_sbmt_cleanup()
is executed, and further processing is abandoned.

==========================================================================*/

void f_sbmt_chaff ()
    /*% f_sbmt_chaff function */

{

/*|
    ## Split Programming Mode -- Host Side ##

    initialize t_sgen_oversamples
    setup data pointer
    use cvmov () to transfer the data of the first chaff bundle from Host to CSPI
use CSPI section to do calculation on this data
FOR each pair of chaff bundles (2 to t_cend_env_def.chaff_count -2 step 2)
LOOP
|
|   ## Using Double-Buffering ##
|   setup data pointer
|   use cvmov () to transfer the data of the current bundle from Host to CSPI
|   use CSPI section to do calculation on this data
|   use cvmov () to transfer the data of the last bundle from CSPI to Host
|
|   setup data pointer
|   use cvmov () to transfer the data of the next bundle from Host to CSPI
|   use CSPI section to do calculation on this data
|   use cvmov () to transfer the data of the current bundle from CSPI to Host
ENDLOOP by chaff bundle

setup data pointer
use cvmov () to transfer the data of the last bundle from Host to CSPI
use CSPI section to do calculation on this data
use cvmov () to transfer this data from CSPI to Host
use xldone () for synchronization

FOR each channel
LOOP
|   add t_sgen_oversamples to t_sgen_aggregate_oversamples
ENDLOOP by channel

## END by Host section ##

## Split Programming Mode -- CSPI Section ##

```

```
    FOR the cell scatters
    LOOP
    |  Compute the point to point range to the chaff bundle
    |  Compute total constant loss
    |  Compute burst receive start time
    |  Generate a sparse chaff bundle sequence
    |  Apply the TxAntennaGain to the sparse sequence
    |  IF ionosphere is "ON" THEN
    |  |  apply ionospheric scintillation to the sparse sequence
    |  ENDIF
    |  apply precipitation attenuation and propagation loss to the
    |  sparse sequence
    |
    |  ## Vectorize antenna -- FOR each antenna  ##
    |  apply RxAntennaGain to sparse sequence
    |  ## END by vectorize antenna ##
    |  ## Vectorize anttenna -- FOR each antenna ##
    |  Determine the sparse sequence of burst receive sample times
    |  Determine the sparse sequence of chaff bundle sample times
    |  and transmit instances
    |  Determine the sequence of waveform samples for the
    |  transmit instances
    |  Determine the full sequence of chaff bundle samples
    |  Determine the chaff bundle return
    |  Add the chaff bundle return to t_sgen_oversamples
    |  FOR each antenna
    |  LOOP
    |  |  ## Vectorize delay tap -- FOR each delay tap ##
    |  |  Determine the sequence of waveform samples for the
    |  |  transmit instances
    |  |  Determine the full sequence of chaff bundle samples
    |  |  Determine the chaff bundle return
    |  |  Add the delay tap return to t_sgen_oversamples
    |  |  ## END by vectorize delay tap ##
    |  ENDLOOP
    |  ## END by vectorize antenna ##
    ENDLOOP by the cell scatters

    ## END by CSPI Section ##
*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

==

================

SUBJECT
PAGE NO. LINE NO. ENCLOSING FUNCTION

NO MARKS TO REFERENCE

```
                          INVOCATION TREE
==========================================================================================
================
PAGE NO.   LINE NO.


   1         50       sbmt_main
  23       1363        | f_sbmt_sparse_orbit
   8        457        | f_sbmt_eclutter
  12        737        | f_sbmt_interpolate
  28       1721        | f_sbmt_precip
  30       1861        | f_sbmt_chaff
   6        315        | f_sbmt_discretes
  26       1581        | f_sbmt_targets
  25       1503        | | f_sbmt_store_int
  16        952        | f_sbmt_jammers
  25       1503        | | f_sbmt_store_int
  19       1135        | f_sbmt_noise
  25       1503        | | f_sbmt_store_int
  21       1257        | f_sbmt_non_linear
  25       1503        | | f_sbmt_store_int
  10        627        | f_sbmt_if_filter
  25       1503        | | f_sbmt_store_int
  14        842        | f_sbmt_iq
  25       1503        | | f_sbmt_store_int
   4        208        | f_sbmt_ad
  25       1503        | | f_sbmt_store_int
  24       1432        | f_sbmt_store

   3        144       i_sbmt_cleanup
```

3-026

ERROR SUMMARY

PAGE NO. LINE NO. ERROR TEXT

NO ERRORS TO REPORT

```
                                                        TBD CROSS REFERENCE REPORT

==========================================================================================================================
===============================
PAGE NO.   LINE NO.           ENCLOSING FUNCTION

   NO TBDs TO REFERENCE
```

E7 RSS Utilities (SUTL) CSU

THIS PAGE INTENTIONALLY LEFT BLANK

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□                    □□□□
□□□□                    □□□□
□□□□                    □□□□
  □□                    □□
  □□       CDADL PROCESSOR       □□
  □□                    □□
  □□                    □□
  □□        VERSION 2.4.6        □□
  □□                    □□
  □□    SERIAL NUMBER: C38U0011    □□
  □□                    □□
  □□      AUTHORIZED USER:       □□
  □□       Raytheon - Canada     □□
  □□                    □□
  □□                    □□
  □□                    □□
  □□                    □□
  □□                    □□
□□□□                    □□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

1 FILE sutl_olog.c
2 FILE sutl_elog.c
4 FILE sutl_clog.c
5 REPORT Requirements Traceability
6 REPORT INVOCATION TREE
7 REPORT ERROR SUMMARY

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sutl_olog.c

Identification: [@(#)]

SCCS Id:        @(#)sutl_olog   1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR    Date        Author    Change made
---  ----------  ------  ------------------------------------
n/a  08-APR-1993   mdb      Initial Release

--------------------------------------------------------------------------

CSU_NAME:       sutl (RSS utilities)

FUNCTION:       f_sutl_olog()

PURPOSE:        This function opens the error log.

The RSS always uses a standard error log file for the
in-progress run.  For playback, this is
specified by c_crpi_spmt_log.  For generation, it is
specified by c_crgi_sgen_log.

If a previous version of the error log exists, it is discarded
and a new version created.

INPUT:          path: pathname of the file to be opened

OUTPUT:         filenum: FILE pointer for the created file
TRUE/FALSE to indicate success/failure of the file creation

=========================================================================*/

#include "sutl_priv.h"

void f_sutl_olog ()
    /*% f_sutl_olog function */

{

/*|

    use fopen() to create/open the log file specified by path

    IF unable to create the file THEN
<-------return(s_spd_failure)
    ELSE
<-------return(s_spd_success)
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sutl_elog.c

Identification: [@(#)]

SCCS Id:        @(#)sutl_elog   1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date         Author   Change made
---  -----------  ------  ----------------------------------------
n/a  08-APR-1993   mdb      Initial Release

--------------------------------------------------------------------------

CSU_NAME:       sutl (RSS utilities)

FUNCTION:       f_sutl_elog()

PURPOSE:        This function writes records to the error log

The RSS always uses a standard error log file for the
in-progress run.  For playback, this is
specified by c_crpi_spmt_log.  For generation, it is
specified by c_crgi_sgen_log.

The message text for the individual error messages is localized
in "errors.h".  Individual modules call this routine with
an error number, and 0-3 integer parameters which are to
be substituted into the text string.

INPUT:          filenum: FILE pointer to the error log
severity: severity to report
error_num: error number to log
params[2]: parameters to include in the error message (optional)

OUTPUT:     TRUE/FALSE to indicate success/failure of the operation

=========================================================================*/

#include "sutl_priv.h"

void f_sutl_elog ()
    /*% f_sutl_elog function */

{
```

```

/*|

    get the current system time
    build error message from time, severity, errors[error_num], params
    write the error message to the stream identified by filenum
    IF error THEN
<-------return(s_spd_failure)
    ELSE
<-------return(s_spd_success)
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=======================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      sutl_clog.c

Identification: [@(#)]

SCCS Id:        @(#)sutl_clog   1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------  ------   -------------------------------------
n/a   08-APR-1993  mdb      Initial Release

---------------------------------------------------------------------

CSU_NAME:       sutl (RSS utilities)

FUNCTION:       f_sutl_clog()

PURPOSE:        This function closes the error log.

The RSS always uses a standard error log file for the
in-progress run.  For playback, this is
specified by c_crpi_spmt_log.  For generation, it is
specified by c_crgi_sgen_log.

INPUT:          filenum: FILE pointer for the created file

OUTPUT:         TRUE/FALSE to indicate success/failure of the file creation

=======================================================================*/

#include "sutl_priv.h"

void f_sutl_clog ()
    /*% f_sutl_clog function */

{

/*|

    use fclose() to close the log file specified by path

    IF unable to close the file THEN
<------return(s_spd_failure)
    ELSE
<-------return(s_spd_success)
    ENDIF

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

SUBJECT
PAGE NO. LINE NO. ENCLOSING FUNCTION

NO MARKS TO REFERENCE

INVOCATION TREE

```
==========================================================================
==============
PAGE NO.   LINE NO.


   4        224      f_sutl_clog

   2        141      f_sutl_elog

   1         51      f_sutl_olog
```

ERROR SUMMARY

PAGE NO. LINE NO. ERROR TEXT

NO ERRORS TO REPORT

```
                                                  TBD CROSS REFERENCE REPORT

==========================================================================================================================
===============================
PAGE NO.   LINE NO.          ENCLOSING FUNCTION

  NO TBDs TO REFERENCE
```

E 312

E8 Playback Manager (SPMT) CSU

THIS PAGE INTENTIONALLY LEFT BLANK

```
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯                    ▯▯▯▯
▯▯▯▯                    ▯▯▯▯
▯▯▯▯                    ▯▯▯▯
  ▯▯                      ▯▯
  ▯▯      CDADL PROCESSOR        ▯▯
  ▯▯                      ▯▯
  ▯▯                      ▯▯
  ▯▯       VERSION 2.4.6       ▯▯
  ▯▯                      ▯▯
  ▯▯    SERIAL NUMBER: C38U0011       ▯▯
  ▯▯                      ▯▯
  ▯▯      AUTHORIZED USER:       ▯▯
  ▯▯      Raytheon - Canada      ▯▯
  ▯▯                      ▯▯
  ▯▯                      ▯▯
  ▯▯                      ▯▯
  ▯▯                      ▯▯
  ▯▯                      ▯▯
▯▯▯▯                    ▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
```

1 FILE spmt.c
3 FILE spmt_initialize.c
4 FILE spmt_tc_hndlr.c
6 FILE spmt_tdit_hndlr.c
8 REPORT Requirements Traceability
9 REPORT INVOCATION TREE
10 REPORT ERROR SUMMARY

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:     spmt.c

Identification: [@(#)]

SCCS Id:       @(#)spmt       1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:      Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   --------------------------------------
n/a  08-APR-1993   mdb      Initial Release

---------------------------------------------------------------------------

CSU_NAME:       spmt (Signal Playback Manager)

FUNCTION:       spmt_main

PURPOSE:        This task starts up the Signal Playback package, then
functions as an RPC listener for the package.  When
the RSS is started by the Testbed Controller, it establishes
contact with the DSP Interface task (tdit), then listens
forever for RPCs from the testbed controller.

spmt also listens for responses from tdit and forwards
status information to the testbed controller.


INPUT                                   SOURCE
-----                                   ------

Start playback command                       Testbed Controller
Abort playback command                        Testbed Controller
Shutdown playback command                      Testbed Controller
Status message                            Testbed Controller
Ping message                              Testbed Controller
Start playback response                    tdit
Abort playback response                     tdit

OUTPUT                                    DESTINATION
------                                  -----------

RPC acknowledge                              Testbed Controller,
tdit
StartSignalPlayback message                tdit
```

```
AbortSignalPlayback message                    tdit

=========================================================================================*/

#include "spmt_priv.h"

void spmt_main()
    /*% Signal Playback Manager Task */

{

/*|

    Use f_spmt_initialize() to initialize the signal playback package

    IF return status != s_spd_success THEN
    |   use exit() to terminate the task; startup has failed
    ENDIF

    t_spmt_rss_state = c_spmt_idle
    t_spmt_run_state = c_spmt_idle

    Use svc_run() to wait for an incoming RPC from the TC or tdit
    ;TC message handler is f_spmt_tc_hndlr()
    ;tdit message handler is f_spmt_tdit_hndlr()
*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}


```

E-3 58

```
/*===========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      spmt_initialize.c

Identification: [@(#)]

SCCS Id:        @(#)spmt_initialize     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date         Author   Change made
---  -----------  ------  ------------------------------------
n/a  08-APR-1993   mdb      Initial Release

-----------------------------------------------------------------------

CSU_NAME:       spmt (Signal Playback Manager)

FUNCTION:       f_spmt_initialize ()

PURPOSE:        This process performs initialization functions for spmt.

===========================================================================*/

void f_spmt_initialize()
    /* f_spmt_initialize function */
{

/*|

    ### initialize RPC interface to testbed controller and tdit ###

    Use pmap_unset() to clean up previous incarnations of the RPC map

    Use svc_register() to register the TCP transport services for the TC and tdit

    Use svctcp_create() to create TCP receiving ports for the TC and tdit

    Use clnt_create() to create a client object for the TC and tdit

    IF no errors reported THEN
<--------return (s_spd_success)
    ELSE
<-------return (s_spd_failure)
    ENDIF

*/
```

```

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local variables */

    /* Code */


}

/* end of spmt_initialize.c */
```

E-260

```
/*=========================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      spmt_tc_hndlr.c

Identification: [@(#)]

SCCS Id:        @(#)spmt_tc_hndlr       1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date         Author   Change made
---   -----------  ------   ---------------------------------------
n/a   08-APR-1993  mdb      Initial Release

-------------------------------------------------------------------------

CSU_NAME:       spmt (Signal Playback Manager)

FUNCTION:       f_spmt_tc_hndlr()

PURPOSE:        This function processes incoming RPCs from the testbed
controller, and forwards them to tdit. It is activated
for each incoming RPC.

=========================================================================*/

void f_spmt_tc_hndlr()
   /*% f_spmt_tc_hndlr() */
{

/*|

   Use svc_getargs() to read the incoming message

   SWITCH on t_crpi_simcmd_types
   |
   |  CASE c_crpi_start_plbcmd
   |  |
   |  |  Use svc_sendreply() to send xdr_void to the TC
   |  |
   |  |  IF t_spmt_rss_state = c_spmt_idle THEN
   |  |  |  t_spmt_rss_state = c_spmt_running
   |  |  |  t_spmt_run_state = c_spmt_running
   |  |  |  use f_sutl_olog(c_crpi_spmt_log...) to initialize error
   |  |  |  log (c_crpi_spmt_log)
   |  |  |  use f_sutl_elog() to log "starting playback run"
   |  |  |  t_spmt_srsdcmd_type = c_spmt_load_playback
| | | t_spmt_playback_parms = t_crpi_playback_parms
| | | use clnt_call() to send message to tdit
| | |
| | ELSE
| | | t_spmt_playback_rsp = c_spmt_start_playback_rsp
| | | status = c_spmt_in_progress
| | | use clnt_call() to send message to tc
| | | use f_sutl_elog() to log "start playback requested with run in
| | | progress"
| | ENDIF
<-------break
| |
| CASE c_crpi_abort_plbcmd
| |
| | Use svc_sendreply() to send xdr_void to the TC
| |
| | IF t_spmt_rss_state is c_spmt_idle THEN
| | | t_spmt_playback_rsp = c_spmt_abort_playback_rsp
| | | status = c_spmt_abort_completed
| | | use clnt_call() to send message to tc
| | ELSE
| | | t_spmt_run_state = c_spmt_aborting
| | | t_spmt_srsdcmd_type = c_spmt_abort_playback
| | | Use clnt_call() to send message to tdit
| | ENDIF
| |
| CASE c_crpi_status_cmd
| |
| | Use svc_sendreply() to send t_spmt_rss_state to the TC
<-------break
| |
    | CASE c_crpi_ping_cmd
    | | Use svc_sendreply() to send xdr_void to the TC
    <-------break
    | |
    | CASE c_crpi_shutdwn_plbcmd
    | |
    | | Use svc_sendreply() to send xdr_void to the TC
    | | t_spmt_run_state = c_spmt_terminating
    | | IF the error log is open THEN
    | | | use f_sutl_elog() to log "playback terminated"
    | | | use f_sutl_clog() to close the error log
    | | ENDIF
    | | t_spmt_srsdcmd_type = c_spmt_abort_playback
    | | Use clnt_call() to send message to tdit
    <-------break
    | |
    ENDSWITCH

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Code */

}
```

```
/*=============================================================================

SPD - Signal Processing Demonstrator Project

(C) Copyright Her Majesty the Queen in Right of Canada as
represented by the Minister of National Defense

File Name:      spmt_tdit_hndlr.c

Identification: [@(#)]

SCCS Id:        @(#)spmt_tdit_hndlr     1.1 93/04/27

Responsible Organization:

Raytheon Canada Limited

Designer:       Marg Baskin

Change History:

SCR   Date        Author   Change made
---   ----------- ------   ----------------------------------------
n/a   08-APR-1993  mdb     Initial Release

------------------------------------------------------------------------

CSU_NAME:       spmt (Signal Playback Manager)

FUNCTION:       f_spmt_tdit_hndlr()

PURPOSE:        This function processes incoming RPCs from tdit,
and forwards them to the testbed controller.

=============================================================================*/

void f_spmt_tdit_hndlr()
    /*% f_spmt_tdit_hndlr() */
{

/*|

    Use svc_getargs() to read the incoming message

    SWITCH on t_spmt_plbrsp_type
    |
    |  CASE c_spmt_playback_ready
    |  |
    |  |  Use svc_sendreply() to send xdr_void to tdit
    |  |  t_spmt_srsdcmd_type = c_spmt_start_playback
    |  |  use clnt_call() to send message to tdit
    <-------break
    |  |
    |  CASE c_spmt_playback_complete
    |  |
    |  |  Use svc_sendreply() to send xdr_void to tdit
    |  |  t_crpi_plbrsp_types = c_crpi_start_plbrsp
    |  |  t_crpi_playback_status = t_spmt_playback_status
```

```
        |  |  use clnt_call() to send the message to the TC
        |  |  use f_sutl_elog() to log "playback complete, iteration %"
        <-------break
        |  |
        |  CASE c_spmt_playback_aborted
        |  |
        |  |  Use svc_sendreply() to send xdr_void to tdit
        |  |  IF t_spmt_run_state = c_spmt_aborting THEN
        |  |  |  use f_sutl_elog() to log "playback aborted"
        |  |  |  use f_sutl_clog() to close the error log
        |  |  |  t_crpi_plbrsp_types = c_crpi_abort_plbrsp
        |  |  |  t_crpi_abort_status = t_spmt_abort_status
        |  |  |  use clnt_call() to send the message to the TC
        |  |  |  t_spmt_rss_state = c_spmt_idle
        |  |  |  t_spmt_run_state = c_spmt_idle
        |  |  <---break
        |  |  ELSE
        |  |  |  use f_sutl_elog() to log "playback terminated"
        |  |  |  use f_sutl_clog() to close the error log
        |  |  |  use exit() to terminate the task
        |  |  ENDIF
        |  |
        ENDSWITCH

    */

        /* Local macros */

        /* Local constants */

        /* Local Structure Definitions */
```

5

```

    /* Local Variables */

    /* Output Messages */

    /* Code */

}

```

E 365

SUBJECT

PAGE NO. LINE NO. ENCLOSING FUNCTION

NO MARKS TO REFERENCE

```
                         INVOCATION TREE
=====================================================================
==============
PAGE NO.  LINE NO.


   1        65      spmt_main
   3       136       |  f_spmt_initialize
   4       211       |  f_spmt_tc_hndlr
   6       332       |  f_spmt_tdit_hndlr
```

ERROR SUMMARY

PAGE NO. LINE NO. ERROR TEXT

NO ERRORS TO REPORT

TBD CROSS REFERENCE REPORT

```
==========================================================================================================
================================
PAGE NO.   LINE NO.          ENCLOSING FUNCTION

   NO TBDs TO REFERENCE
```

E :7D

E9 DSP Interface (TDIT) CSU

THIS PAGE INTENTIONALLY LEFT BLANK

DSP Interface CSC/CSU Design Logic and Applicable Reports

```
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
[][][][]                                            [][][][]
[][][][]                                            [][][][]
[][][][]                                            [][][][]
  [][]                                                [][]
  [][]              CDADL PROCESSOR                   [][]
  [][]                                                [][]
  [][]                                                [][]
  [][]               VERSION 2.4.6                    [][]
  [][]                                                [][]
  [][]           SERIAL NUMBER: CJ8U0011              [][]
  [][]                                                [][]
  [][]             AUTHORIZED USER:                   [][]
  [][]            Raytheon - Canada                   [][]
  [][]                                                [][]
  [][]                                                [][]
  [][]                                                [][]
  [][]                                                [][]
  [][]                                                [][]
[][][][]                                            [][][][]
[][][][][][][][][][][][][][][][][][][][][][][][][][][][][][]
```

```
 1      FILE tdit_main.c
 3      FILE tdit_initialize.c
 5      FILE tdit_abort_playback.c
 6      FILE tdit_start_playback.c
 7      FILE tdit_time_iteration.c
 8      REPORT TBD CROSS REFERENCE REPORT
 9      REPORT INVOCATION TREE
10      REPORT ERROR SUMMARY
```

```
/*%%=================================================================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 1992

File Name:      tdit_main.c

Identification: [@(#)]

SCCS Id:        @(#)tdit_main.c 1.2 93/06/14

Responsible Organization:

Raytheon Canada Limited

Designer:       Glenn Price

Change History:

SCR  Date        Author   Change made
---  ----------  ------   ---------------------------------------------------------------------------------

CSU_NAME:   DSP Interface

FUNCTION:   f_tdit_dsp_interface

PURPOSE:    The main processing thread of DSP Interface which recieves and executes SRSD Commands from DSPI RPC Server and controls
the overall operation of the RTSG

INPUT                               SOURCE
-----                               ------
srsd_command                        DSPI RPC Server
HOST_ITER_COMPLETE_EVENT            DSP Output

OUTPUT                              DESTINATION
------                              -----------
acknowledge                         DSPI RPC Server
error_bad_command                   DSPI RPC Server

CREATION DATE:  May 17 1993

=================================================================================================================*/

#include "tdit.h"

int f_tdit_dsp_interface()
    /*% f_tdit_dsp_interface */

{

/*!

    err_code = f_tdit_initialize()

    do
    |
    |   <Wait for an incoming message>
    |
    |   crq_status = msgQReceive( tdit_srsd_command_que, &srsd_command, sizeof( srsd_command )(sic), NO_WAIT )
    |
    |   if( crq_status != ERROR ) then
    |   |   signal_status = msgQReceive( tdit_vai_signal_que, &signal_message, sizeof( signal_message )(sic), NO_WAIT )
    |   else
    |   |   signal_status = msgQReceive( tdit_vai_signal_que, &signal_message, sizeof( signal_message )(sic), 1/2 second )
    |   endif
    |
    |   <case sbmt task>
    |   if( crq_status != ERROR ) then
    |   |
    |   |   if( srsd_command->command_type is one of LoadPlayback, AbortPlayback or StartPlayback ) then
    |   |   |   <acknowledge the message>
    |   |   |   make an acknowledge buffer
    |   |   |   err_code = msgQSend( trsv_reponse_que, acknowledge buffer, sizeof( acknowledge buffer ), NO_WAIT, MSG_PRI_NORMAL )
    |   |   else
    |   |   |   create an error_bad_command buffer
    |   |   |   err_code = msgQSend( trsv_reponse_que, error_bad_command buffer, sizeof( error_bad_command buffer ), NO_WAIT,
    |   |   |   MSG_PRI_NORMAL )
    |   |   endif
    |   |
    |   |   switch srsd_command->command_type
    |   |   |   case LoadPlayback from spmt
    |   |   |   |
    |   |   |   |   err_code = f_tdit_load_for_playback()
    |   |   |   |
    |   |   <--------break
    |   |   |   |
    |   |   |   case of AbortPlayback from spmt
    |   |   |   |
    |   |   |   |   err_code = f_tdit_abort_playback()
    |   |   |   |
    |   |   <--------break
    |   |   |   |
    |   |   |   case of StartPlayback from spmt
    |   |   |   |
    |   |   |   |   err_code = f_tdit_start_playback()
    |   |   |   |
    |   |   <--------break
    |   |   |   |
    |   |   endswitch
    |   |
    |   endif
```

f 2-4

```
          |
          |    if( signal_status != ERROR ) then
          |    |    if( mode == RUNNING ) then
          |    |    |
          |    |    |    err_code = f_tdit_time_iteration()
          |    |    else
          |    |    |    error_report( timing_signal_recieved_while_not_in_run_mode )
          |    |    endif
          |    |
          |    endif
          |
          dowhile forever
          enddo

     */

     }
```

```
/*%%===========================================================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 1992

File Name:      tdit_initialize.c

Identification: [@(#)]

SCCS Id:        @(#)tdit_initialize.c   1.1 93/06/14

Responsible Organization:

Raytheon Canada Limited

Designer:       Glenn Price

Change History:

SCR   Date        Author   Change made
---   ----------  ------   -----------------------------------------------------------------------

CSU_NAME:   DSP Interface

FUNCTION:   f_tdit_initialize

PURPOSE:    Initializes the rpc clnt handle and the message queues to be used by this CSU

INPUT                            SOURCE
-----                            ------

OUTPUT                           DESTINATION
------                           -----------
spmt_handle                      (Global Data)
tdit_srsd_command_que            (Global Data)
trsv_reponse_que                 (Global Data)
tdit_vai_srq_que                 (Global Data)
tdit_vai_response_que            (Global Data)
tdit_vai_signal_que              (Global Data)
tdit_vai_error_que               (Global Data)

CREATION DATE:  May 17 1993

===========================================================================================================*/

#include "tdit.h"

int f_tdit_initialize()
    /*% f_tdit_initialize */

{

/*|

    <initialize interfaces to spmt, tiop>
    # create a handle to spmt server #
    spmt_handle = clnt_create( spmt_server, ...., ...., "tcp" )

    # initialize interfaces to tdit's rpc server #

    # create tdit_srsd_commands_que, the queue used by rcp server to to send srsd_commands to DSP interface #
    tdit_srsd_command_que = msgQCreate( enough, enough, MSG_Q_FIFO )

    # we create the srsv_response que here because it is not easily done by the server #
    trsv_reponse_que = msgQCreate( enough, enough, MSG_Q_FIFO )

    # create tdit_vai_srq_que, the service request queue used by the IOP's to transfer the radar data file
    tdit_vai_srq_que = msgQCreate( enough, enough, MSG_Q_FIFO )

    # create tdit_vai_response_que, the queue that VAI will use to send the PE responses to DSP Interface #
    tdit_vai_response_que = msgQCreate( enough, enough, MSG_Q_FIFO )

    # connect the data service queue to a VAI event_flag #
    ioctl( VAI_QID_EF_TABLE, VAI_QCONNECT, &qid_ef_entry_for_tdit_vai_srq_que )

    # create tdit_vai_signal_que, the queue used by all the PE's to signal to DSP Interface #
    tdit_vai_signal_que = msgQCreate( enough, enough, MSG_Q_FIFO )

    # connect the signal service queue to a VAI event_flag #
    ioctl( VAI_QID_EF_TABLE, VAI_QCONNECT, &qid_ef_entry_for_tdit_vai_signal_que )

    # create tdit_vai_error_que, the queue used by all the PE's to status messages to DSP Interface #
    tdit_vai_error_que = msgQCreate( enough, enough, MSG_Q_FIFO )

    # connect the error service queue to a VAI event_flag #
    ioctl( VAI_QID_EF_TABLE, VAI_QCONNECT, &qid_ef_entry_for_tdit_vai_error_que )

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Codes */

```

```
  }
```

E-277

```
/*%%==========================================================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 1992

File Name:      tdit_abort_playback.c

Identification: [@(#)]

SCCS Id:        @(#)tdit_abort_playback.c   1.2 93/06/14

Responsible Organization:

Raytheon Canada Limited

Designer:       Glenn Price

Change History:

SCR   Date       Author   Change made
---   --------   ------   ---------------------------------------------------------------

CSU_NAME:   DSP Interface

FUNCTION:   f_tdit_abort_playback

PURPOSE:    Tell DSP Output to abort its playback

INPUT                          SOURCE
-----                          ------
mode                           (Global Data)

OUTPUT                         DESTINATION
------                         -----------
PE_REFRESH_EVENT               DSP Output
PlaybackAborted                DSPI RPC Server
mode                           (Global Data)

CREATION DATE:  May 17 1993

==========================================================================================================*/

#include "tdit.h"

int f_tdit_abort_playback()
    /*% f_tdit_abort_playback */

{

/*|

    <if playback is currently in progress _then>
    if( mode == RUNNING ) then
    |   <issue a Refresh to tiop to reset the data pointers to the beginning of the buffer, in preparation for a resumption of
    |   playback>
    |
    |   initialize a vai_command_request struct with a host to pe event flag command for PE_REFRESH_EVENT
    |
    |   for( pe_num = 0; pe_num < NUM_IOPS; pe_num++ and err_response != ERROR; pe_num++ )
    |   loop
    |   |   write( pe[ pe_num ].fd, vai_command_request .... )
    |   |
    |   |   err_response = msgQRecieve( tdit_vai_response_que, &response_message .... WAIT_FOREVER )
    |   |
    |   endloop
    |
    |   # check for errors on send PE_REFRESH_EVENT #
    |   if( err_response == ERROR ) then
    |   |   error_report( send_refresh_event_error )
    |   |   mode = RUNNING;
    |   else
    |   |   mode = INITIALIZED;
    |   endif
    |
    else
    |   error_report( attempt_to_abort_while_not_running )
    endif

    <send PlaybackAborted to spmt>
    make an PlaybackAborted buffer
    rpc_call( smpt_handle, PlaybackAborted )

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Codes */

}
```

```
/*%%=============================================================================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 1992

File Name:      tdit_start_playback.c

Identification: [@(#)]

SCCS Id:        @(#)tdit_start_playback.c   1.2 93/06/14

Responsible Organization:

Raytheon Canada Limited

Designer:       Glenn Price

Change History:

SCR    Date         Author    Change made
---    ------------------------------------------------------------------------------------------

CSU_NAME:   DSP Interface

FUNCTION:   f_tdit_start_playback

PURPOSE:    Tell DSP Output to start sending SRSD Data to the DSP

INPUT                              SOURCE
-----                              ------
mode                               (Global Data)

OUTPUT                             DESTINATION
------                             -----------
PE_GO_EVENT                        DSP Output
mode                               (Global Data)

CREATION DATE:  May 17 1993

===============================================================================================================================*/

#include "tdit.h"

int f_tdit_start_playback()
    /*% f_tdit_start_playback */

{

/**)

    if( mode == INITIALIZED ) then
    |
    |   # G. Price changed this requirement logic statement #
    |   # we let the iops send each iteration on its own #
    |   <send a Go command to tiop, telling it to transmit iteration_count iterations of the RTSC data to the ESP>
    |
    |   initialise a vai_command_request struct with a host to pe event flag command for PE_GO_EVENT
    |
    |   for( pe_num = 0; pe_num < NUM_IOPS; pe_num++ and err_response != ERROR; pe_num++ )
    |   loop
    |   |   write( pe[ pe_num ].fd, vai_command_request .... )
    |   |
    |   |   err_response = msgQRecieve( tdit_vai_response_que, &response_message .... WAIT_FOREVER )
    |   |
    |   endloop
    |
    |   # check for errors on send PE_GO_EVENT #
    |   if( err_response == ERROR ) then
    |   |   error_report( send_go_event_error )
    |   |   mode = NOT_RUNNING;
    |   else
    |   |   mode = RUNNING;
    |   |
    |   |   # reset tick count to 0 for the first iteration #
    |   |   tickSet( 0 )
    |   |
    |   |   iteration_count = 0
    |   |
    |   endif
    |
    else
    |   error_report( attempt_to_run_while_not_initialized )
    endif

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Codes */

}
```

E

```
/*%%=====================================================================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 1992

File Name:      tdit_time_iteration.c

Identification: [@(#)]

SCCS Id:        @(#)tdit_time_iteration.c   1.1 93/06/14

Responsible Organization:

Raytheon Canada Limited

Designer:       Glenn Price

Change History:

SCR   Date         Author    Change made
---   ----------   ------    ----------------------------------------------------------

CSU_NAME:   DSP Interface

FUNCTION:   f_tdit_time_iteration

PURPOSE:    Recieves and executes SRSD Commands from Playback Manager and controls the overall operation of the RTSG

INPUT                          SOURCE
-----                          ------
signal_message                 (Global Data)

OUTPUT                         DESTINATION
------                         -----------
iteration_count                (Global Data)
PlaybackComplete               (Playback Manager)

CREATION DATE:  May 17 1993

=====================================================================================================================*/

#include "tdit.h"

int f_tdit_time_iteration()
    /*% f_tdit_time_iteration */

{

/*|

    if( signal_message.hostEventFlag == HOST_ITER_COMPLETE_EVENT ) then
    |   tick_count = tickGet()
    |
    |   # reset tick count to 0 for next iteration #
    |   tickSet( 0 )
    |
    |   usign tick_count and system set tick duration calculate the iteration_time for the iteration
    |
    |   -send PlaybackComplete message to spmt, giving timestamp and iteration number-
    |
    |   iteration_count++
    |
    |   -send PlaybackComplete message to spmt, giving timestamp and iteration number-
    |   make a PlaybackComplete with iteration_time and iteration_count
    |
    |   rpc_call( smpt_handle, PlaybackComplete )
    |
    endif

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Codes */

}
```

TBD CROSS REFERENCE REPORT

```
==========================================================================================================
PAGE NO.   LINE NO.          ENCLOSING FUNCTION

   NO TBDs TO REFERENCE
```

```
                                         INVOCATION TREE

==========================================================================================================================
=
PAGE NO.   LINE NO.

   1          47        f_tdit_dsp_interface
   3         173        |    f_tdit_initialize
   5         275        |    f_tdit_abort_playback
   6         374        |    f_tdit_start_playback
   7         476        |    f_tdit_time_iteration
```

```
                                                          PAGE 10
                         ERROR SUMMARY
=========================================================================================
PAGE NO.   LINE NO.    ERROR TEXT

  NO ERRORS TO REPORT
```

E-2-1

```
                                                  TBD CROSS REFERENCE REPORT
========================================================================================================================
================================
PAGE NO.   LINE NO.         ENCLOSING FUNCTION

  NO TBDs TO REFERENCE
```

E-2-24

E10 DSP Output (TIOP) CSU

THIS PAGE INTENTIONALLY LEFT BLANK

DSP Output CSC/CSU Design Logic and Applicable Reports

```
[][][][][][][][][][][][][][][][][][][][][][][][][][]
[][][][]                                    [][][][]
[][][][]                                    [][][][]
[][][][]                                    [][][][]
  [][]                                        [][]
  [][]            CDADL PROCESSOR             [][]
  [][]                                        [][]
  [][]                                        [][]
  [][]             VERSION 2.4.6              [][]
  [][]                                        [][]
  [][]        SERIAL NUMBER: C38U0011         [][]
  [][]                                        [][]
  [][]           AUTHORIZED USER:             [][]
  [][]          Raytheon - Canada             [][]
  [][]                                        [][]
  [][]                                        [][]
  [][]                                        [][]
  [][]                                        [][]
  [][]                                        [][]
[][][][]                                    [][][][]
[][][][][][][][][][][][][][][][][][][][][][][][][][]

1       FILE tiop_main.c
3       FILE tiop_playback.c
5       FILE tiop_load_new_run.c
7       REPORT TBD CROSS REFERENCE REPORT
8       REPORT INVOCATION TREE
9       REPORT ERROR SUMMARY
```

E·687

```
/*%%=====================================================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 1992

File Name:      tiop_main.c

Identification: [@(#)]

SCCS Id:        @(#)tiop_main.c 1.3 93/06/14

Responsible Organization:

Raytheon Canada Limited

Designer:       Glenn Price

Change History:

SCR   Date         Author   Change made
---   ----------   ------   ----------------------------------------------------------------

CSU_NAME:   DSP Output

FUNCTION:   f_tiop_dsp_output

PURPOSE:    The main processing thread of DSP Output which recieves control signals from DSP Interface acting on them
accordingly. eg given a PE_RESET_EVENT signel DSP Output will initiate the downloading of run parameters as well
as srsd data for a subsequent run.

INPUT                           SOURCE
-----                           ------
PE_GO_EVENT                     DSP Interface
PE_RESET_EVENT                  DSP Interface

OUTPUT                          DESTINATION
------                          -----------
control_flag                    (Global Data)

CREATION DATE:  May 18 1993

=======================================================================================================*/

#include "tiop.h"

int f_tiop_dsp_output()
    /*% f_tiop_dsp_output */

{

/*!
    # note: <..>'s or ##'s denote comments on the detailed design
    $ denotes a variable name as specified in DSP Parameters table, non $ variables denote ESP derived names #

    <initialize interfaces to tdit>

    <initialize data interfaces>
    err_spb = SPB_initialize_f( ... ) dsp interface bus one way transfer (outgoing) 32 bit mode

    also init input TPMs
    err_sbi = SBI_init_TPM_f( ... )
    err_sbi != SBI_init_TPM_f( ... )

    if( err_sbi != OK or err_spb != OK ) then
    |   error_report( tiop_init )
    endif

    for( ;; )
    loop
    |
    |   #check for any process control signals from DSP Interface#
    |   err_go_ef = ef_test_and_clear_f( PE_GO_EVENT )
    |   err_reset_ef = ef_test_and_clear_f( PE_RESET_EVENT )
    |
    |   if( err_go_ef == EF_WAS_SET_c ) then
    |   |
    |   |   err_code = f_tiop_playback()
    |   |
    |   endif
    |
    |   if( err_reset_ef == EF_WAS_SET_c ) then
    |   |   control_flag = NEW_RUN
    |   |
    |   |   err_code = f_tiop_load_new_run()
    |   |
    |   endif
    |
    endloop

 */

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */
```

```
        /* Codes */

    }
```

E-229

```
/*%%=================================================================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 1992

File Name:       tiop_playback.c

Identification: [@(#)]

SCCS Id:         @(#)tiop_playback.c 1.2 93/06/14

Responsible Organization:

Raytheon Canada Limited

Designer:        Glenn Price

Change History:

SCR   Date         Author    Change made
---   ----------   ------    -----------------------------------------------------------------------------------

CSU_NAME:   DSP Output

FUNCTION:   f_tiop_playback

PURPOSE:    Given a go signal from DSP Interface begin sending srsd data to the DSP. f_tiop_playback will continually send
srsd data until a PE_REFRESH_EVENT is received or until iteration_count runs of the data have been sent to the DSP

INPUT                             SOURCE
-----                             ------
run_params                        (Global Data)
srsd_data                         SRSD Store
PE_REFRESH_EVENT                  DSP Interface

OUTPUT                            DESTINATION
------                            -----------
srsd_data                         DSP
HOST_ITER_COMPLETE_EVENT          DSP Interface
control_flag                      (Global Data)

CREATION DATE:  May 18 1993

==================================================================================================================*/

#include "tiop.h"

int f_tiop_playback()
    /*% f_tiop_playback */

{

/*|

    control_flag = GO

    for( iter_count = 0; ( iter_count < iteration_count ) && ( control_flag != REFRESH ); iter_count++ )
    loop
    |   for( nci_count = 0, cpi_count = 0, next_cpi_offset = 0; ( nci_count < tot_ncis_in_run ) &&
    |   |   ( control_flag != REFRESH ); nci_count++ )
    |   loop
    |   |
    |   |   for( cur_cpi_in_nci = 0; ( cur_cpi_in_nci < tot_cpis_in_cur_nci[ nci_count ] ) && ( control_flag != REFRESH );
    |   |   |   cur_cpi_in_nci++, cpi_count++ )
    |   |   loop
    |   |   |
    |   |   |   determine start_range_bin and end_range_bin, the range of range bins for the first CPI section given IOP
    |   |   |   number, NUM_IOPS and pri_size[ nci_count ]
    |   |   |
    |   |   |   start_range_bin = floor( ( IOP_NUM - 1 ) * pri_size[ nci_count ] / NUM_IOPS )
    |   |   |   end_range_bin = floor( IOP_NUM * pri_size[ nci_count ] / NUM_IOPS - 1 )
    |   |   |
    |   |   |   if( IOP_NUM == NUM_IOPS ) then
    |   |   |   |   end_range_bin = pri_size[ nci_count ] - 1
    |   |   |   endif
    |   |   |
    |   |   |   determine num_buffers required to transfer the range of range gates for all the pulses for all the
    |   |   |   channels in the config
    |   |   |   section_size = ( end_range_bin - start_range_bin + 1 ) * longwords_per_sample * num_channels
    |   |   |   (longword is 32-bits)
    |   |   |
    |   |   |   num_pulses_txfrd = ceil( $num_samp_input[ cpi_count ] / pri_size[ nci_count ] )
    |   |   |
    |   |   |   num_buffers = section_size * num_pulses_txfrd / MAX_SPB_BUF_SIZE
    |   |   |
    |   |   |   tpm_buf_offset = next_cpi_offset + start_range_bin * num_pulses_txfrd * num_channels * longwords_per_sample
    |   |   |
    |   |   |   next_cpi_offset += pri_size[ nci_count ] * $num_samp_input[ cpi_count ] * num_channels *
    |   |   |   longwords_per_sample
    |   |   |
    |   |   |   for num_buffers
    |   |   |   loop
    |   |   |   |
    |   |   |   |   do
    |   |   |   |   |   err_spb = SPB_tx_check_buf_empty_f( port, &buffer )
    |   |   |   |   dowhile( err_spb != SPB_idle_e or spb errors )
    |   |   |   |   enddo
    |   |   |   |
    |   |   |   |   if( err_spb == SPB_idle_e ) then
    |   |   |   |   |   load tpm_ptr[ tpm_buf_offset ... tpm_buf_offset + MAX_SPB_BUF_SIZE ] to buffer
    |   |   |   |   |
|   |   |   |   |   tpm_buf_offset = tpm_buf_offset + MAX_SPB_BUF_SIZE
|   |   |   |   |
|   |   |   |   |   send the ersd buffer
|   |   |   |   |   err_spb = SPB_tx_set_buf_full( port, MAX_SPB_BUF_SIZE )
|   |   |   |   |
|   |   |   |   endif
|   |   |   |
|   |   |   endloop
|   |   |
|   |   |   # send last fractional spb buffer if there is one #
|   |   |   if( ( section_size * num_pulses_txfrd ) modulo MAX_SPB_BUF_SIZE ) != 0 ) then
|   |   |   |   num_buffers++
|   |   |   |   do
|   |   |   |   |   err_spb = SPB_tx_check_buf_empty_f( port, &buffer )
|   |   |   |   dowhile( err_spb != SPB_idle_e or spb errors )
|   |   |   |   enddo
|   |   |   |
|   |   |   |   fract_spb_buf_length = ( section_size * num_pulses_txfrd ) modulo MAX_SPB_BUF_SIZE
|   |   |   |
|   |   |   |   if( err_spb == SPB_idle_e ) then
|   |   |   |   |   load tpm_ptr[ tpm_buf_offset ... tpm_buf_offset + fract_spb_buf_length ] to buffer
|   |   |   |   |
|   |   |   |   |   tpm_buf_offset = tpm_buf_offset + fract_spb_buf_length
|   |   |   |   |
|   |   |   |   |   send the ersd buffer
|   |   |   |   |   err_spb = SPB_tx_set_buf_full( port, fract_spb_buf_length )
|   |   |   |   |
|   |   |   |   endif
|   |   |   |
|   |   |   endif
|   |   |
|   |   |   err_refresh_ef = ef_test_and_clear_f( PE_REFRESH_EVENT )
|   |   |
|   |   |   if( err_refresh_ef == EF_WAS_SET_c ) then
|   |   |   |   control_flag = REFRESH
|   |   |   endif
|   |   |
|   |   endloop
|   |
|   endloop
|
|   # send an iteration complete signal only if were the last iop #
|   if( iop_num == NUM_IOPS ) then
|   |   err_code = VME_signal_serv_req_f( HOST_ITER_COMPLETE_EVENT )
|   endif
|
endloop

*/

    /* Local macros */

    /* Local constants */

    /* Local Structure Definitions */

    /* Local Variables */

    /* Output Messages */

    /* Codes */

}
```

```
/*&&==========================================================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 1992

File Name:       tiop_load_new_run.c

Identification: [@(#)]

SCCS Id:         @(#)tiop_load_new_run.c 1.2 93/06/14

Responsible Organization:

Raytheon Canada Limited

Designer:        Glenn Price

Change History:

SCR   Date         Author   Change made
---   ----------   ------   -----------------------------------------------------------------------------

CSU_NAME:   DSP Output

FUNCTION:   f_tiop_load_new_run

PURPOSE:    Get the run parameters and srsd data from DSP Interface and put the srsd data to the TPM's (srsd store). The
SRSD Data is stored as one big array in [range_bin][pulse][channel_num] order.

INPUT                                 SOURCE
-----                                 ------
PE_CONFIG_EVENT                       DSP Interface
run_params                            DSP Interface
PE_GET_DATA_EVENT                     DSP Interface
srsd_buffer                           DSP Interface

OUTPUT                                DESTINATION
------                                -----------
HOST_CONFIG_EVENT                     DSP Interface
HOST_GET_DATA_EVENT                   DSP Interface
srsd_buffer.data                      SRSD Store
control_flag                          (Global Data)

CREATION DATE:  May 18 1993

==============================================================================================================*/

#include "tiop.h"

int f_tiop_load_new_run()
    /*% f_tiop_load_new_run */

{

/*!

    control_flag = NEW_RUN

    # get the run params from DSP Interface #
    err_code = VME_data_serv_req_f( HOST_CONFIG_EVENT, VME_PUT_e, ... PE_CONFIG_EVENT, run_params, sizeof( run_params ) )
    do
    |   err_config_ef = EF_test_and_clear( PE_CONFIG_EVENT )
    dowhile( err_config_ef != EF_WAS_SET_C )
    enddo

    # run_params are now valid for the run #

    if( iop_num == INPUT_IOP1 ) then
    |   tpm_ptr = base address of the TPM
    |
    |   for( nci_count = 0, cpi_count = 0; nci_count < tot_ncis_in_run; nci_count++ )
    |   loop
    |   |
    |   |   for( cur_cpi_in_nci = 0; cur_cpi_in_nci < tot_cpis_in_cur_nci[ nci_count ]; cur_cpi_in_nci++, cpi_count++ )
    |   |   loop
    |   |   |
    |   |   |   num_pulses_txfrd = ceil( $num_samp_input[ cpi_count ] / pri_size[ nci_count ] )
    |   |   |
    |   |   |   reads_required = ceil( num_pulses_txfrd / NUM_PULSES_PER_READ )
    |   |   |
    |   |   |   # read a cpi of data #
    |   |   |   for( last_pulse = 0, num_reads = 0; num_reads < reads_required; num_reads++ )
    |   |   |   loop
    |   |   |   |   # get a partial corner turned radar buffer from DSP Interface eg tell him to put a radar buffer to our
    |   |   |   |   expansion memory #
    |   |   |   |   err_code = VME_data_serv_req_f( HOST_GET_DATA_EVENT, VME_PUT_e, ... PE_GET_DATA_EVENT, srsd_buffer,
    |   |   |   |   pri_size[ nci_count ] * num_channels * NUM_PULSES_PER_READ )
    |   |   |   |
    |   |   |   |   do
    |   |   |   |   |   err_config_ef = EF_test_and_clear( PE_GET_DATA_EVENT )
    |   |   |   |   dowhile( err_config_ef != EF_WAS_SET_C )
    |   |   |   |   enddo
    |   |   |   |
    |   |   |   |   # put srsd_buffer.data the TPM's using srsd_buffer.num_pulses and srsd_buffer.pri_size in pulse order #
    |   |   |   |   # data will be stored in [range_gate][pulse][channel_num] order #
    |   |   |   |
    |   |   |   |   for( range_gate = 0; range_gate < srsd_buffer.pri_size; range_gate++ )
    |   |   |   |   loop
    |   |   |   |   |
    |   |   |   |   |   for( pulse_num = last_pulse, count_pulse = 0; count_pulse < srsd_buffer.num_pulses; count_pulse++,
    |   |   |   |   |   |   pulse_num++ )
```

E-1-2

```
        |    |    |    |    |    loop
        |    |    |    |    |    |
        |    |    |    |    |    |    tpm_ptr[ range_gate ][ pulse_num * num_pulses_txfrd ][ 0 ... num_channels ] =
        |    |    |    |    |    |    ersd_buffer.data[ range_gate ][ count_pulse ][ 0 ... num_channels ]
        |    |    |    |    |    |
        |    |    |    |    |    endloop
        |    |    |    |    |
        |    |    |    |    endloop
        |    |    |    |
        |    |    |    |    last_pulse = last_pulse + ersd_buffer.num_pulses
        |    |    |    |
        |    |    |    endloop
        |    |    |
        |    |    endloop
        |    |
        |    endloop
        |
        endif

    */

        /* Local macros */

        /* Local constants */

        /* Local Structure Definitions */

        /* Local Variables */

        /* Output Messages */

        /* Codes */

    }
```

```
                                   TBD CROSS REFERENCE REPORT
================================================================================================================

PAGE NO.   LINE NO.          ENCLOSING FUNCTION

  NO TBDs TO REFERENCE
```

[illegible]

```
==========================================================================================================================================
=
PAGE NO.   LINE NO.

    1          47        f_tiop_dmp_output
    3         154        |    f_tiop_playback
    5         323        |    f_tiop_load_new_run
```

E-2-5

```
                                        ERROR SUMMARY
==========================================================================================================
PAGE NO.   LINE NO.    ERROR TEXT

  NO ERRORS TO REPORT
```

E [illegible]

```
                                                           TBD CROSS REFERENCE REPORT
=================================================================================================================
==============================
PAGE NO.   LINE NO.          ENCLOSING FUNCTION

   NO TBDs TO REFERENCE
```

THIS PAGE INTENTIONALLY LEFT BLANK

E11 RPC Server (TRSV) CSU

THIS PAGE INTENTIONALLY LEFT BLANK

RPC Server CSC/CSU Design Logic and Applicable Reports

```
[][][][][][][][][][][][][][][][][][][][][][][][][][][]
[][][][]                                      [][][][]
[][][][]                                      [][][][]
[][][][]                                      [][][][]
  [][]                                          [][]
  [][]             CDADL PROCESSOR              [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]              VERSION 2.4.6               [][]
  [][]                                          [][]
  [][]         SERIAL NUMBER: C38U0011          [][]
  [][]                                          [][]
  [][]             AUTHORIZED USER:             [][]
  [][]            Raytheon - Canada             [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]                                          [][]
  [][]                                          [][]
[][][][]                                      [][][][]
[][][][][][][][][][][][][][][][][][][][][][][][][][][]

1       FILE trav_main.c
2       REPORT TBD CROSS REFERENCE REPORT
3       REPORT INVOCATION TREE
4       REPORT ERROR SUMMARY
```

```
/*%%=====================================================================================================================

SPD - Signal Processing Demonstrator Computer System

(C) Copyright Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 1992

File Name:      trsv_main.c

Identification: [@(#)]

SCCS Id:        %W% %E%

Responsible Organization:

Raytheon Canada Limited

Designer:       Glenn Price

Change History:

SCR   Date        Author    Change made
---  ----------  ------  ---------------------------------------------------------------------------

CSU_NAME:   DSPI RPC Server

FUNCTION:   f_trsv_dspi_rpc_server

PURPOSE:    Acts as a server for DSP Interface. It simply passes srsd_commands onto DSP Interface who then acts on those commands.
A separate server is necessary to decouple DSP Interface from the closely coupled interface of the server.

INPUT                           SOURCE
-----                           ------
srsd_command                    Playback Manager
acknowledge                     DSP Interface

OUTPUT                          DESTINATION
------                          -----------
srsd_command                    DSP Interface
acknowledge                     Playback Manager

CREATION DATE:  May 20 1993

====================================================================================================================*/

#include    "trsv.h"

f_trsv_dspi_rpc_server()
    /*% f_trsv_dspi_rpc_server */
{

/*|

    err_code = msgQSend( tdit_srsd_command_que, srsd_command, sizeof( srsd_command ), WAIT_FOREVER,
    MSG_PRI_NORMAL )

    if( ( srsd_command == LoadPlayback ) or ( srsd_command == AbortPlayback ) or ( srsd_command == StartPlayback ) ) then
    |   err_code = msgQReceive( trsv_reponse_que, acknowledge, sizeof( acknowledge )(sic), WAIT_FOREVER )
    |
    |   check acknowledge message
    |
<--------return( acknowledge )
    else
<--------return( void )
    endif

*/

}
```

E-302

```
                                        TBD CROSS REFERENCE REPORT
=================================================================================================
PAGE NO.   LINE NO.          ENCLOSING FUNCTION

    NO TBD# TO REFERENCE
```

```
                                                  INVOCATION TREE                                      PAGE 3
=============================================================================================================================
PAGE NO.   LINE NO.

    1          47        f_trav_dspi_rpc_server
```

```
                                                    ERROR SUMMARY                                                   PAGE 4
==========================================================================================================================
PAGE NO.   LINE NO.    ERROR TEXT

  NO ERRORS TO REPORT
```

E-73

```
                                                   TBD CROSS REFERENCE REPORT
========================================================================================================================
================================
PAGE NO.   LINE NO.         ENCLOSING FUNCTION

  NO TBDs TO REFERENCE
```

MACDONALD DETTWILER
RAYTHEON

Ref: SB-DS-50-4059
Issue/Revision: 1/0
Date: SEPT. 30, 1993

APPENDIX F

RTSG HWCI DESIGN DOCUMENTATION LIBRARY

By: J. Graaskamp (RCL)

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

Page

APPENDIX F **RTSG HWCI DESIGN DOCUMENTATION LIBRARY**

APPENDIX F RTSG HWCI DESIGN DOCUMENTATION LIBRARY F-1
F1 Type DEFS and Global Data F-1
F2 Signal Generation Manager (SSGM) CSU F-5

APPENDIX F
RTSG HWCI DESIGN DOCUMENTATION LIBRARY

F1 **Parts List**

RTSG Parts List	PLG442051 (2 pages)

THIS PAGE INTENTIONALLY LEFT BLANK

SBR PRELIMINARY PARTS LIST

TITLE: Real Time Signal Generator		PLG442051				
REVISION 4		DESCRIPTION				DATE 5 November 1993
FIND NO.	QTY	PART NUMBER	DWG CODE	MIL OR IND SPEC NO.	DOC TYPE	TITLE
1	REF	G442051				Assy, Real Time Signal Generator
2	REF	SDG442051				Schematic, Real Time Signal Generator
3	1	G442055				Card Cage, 20 Slot, 2 VME, 18 VASP
						Manufacturer, Mupac
4	1	G442052				Backplane, 20 Slot Custom, 2 VME, 18 VASP
						MDA P/N 23-9731-R00
5	3	600-00966			VPN	CCA, IOP
						A4,A19
						Manufacturer, Spectrum
6	1	MVME147P2			VPN	CCA, ADAPTER
						Manufacturer, Motorola
7	6	600-00984			VPN	CCA, TPM
						A5,A6,A7,A16,A17,A18
						Manufacturer, Spectrum
8	3	23-9729-R00			VPN	CCA, IOP SPB ADAPTER
						Manufacturer, MDA
9	1	MVME167-32			VPN	CCA, Single Board Computer, 8 Meg, 33 MHz
						A1
						Manufacturer, Motorola
10	1	MVME712A			VPN	CCA, Transiton Card, Serial Ports
						Manufacturer, Motorola
11	1	MVME712KC			VPN	CCA, Transition Card, Ethernet
						Manufacturer, Motorola
12	14	PV10-6R-L			VPN	Terminal Lug, Insulated, #12-10 Wire
						Manufacturer Panduit
13	4	622-50SM1			VPN	Connector,D-Type, Metal Shell, 50 Sockets
						Manufacturer T&B/Ansley
						IOP 1 SPB H,L; IOP 3 SPB H,L; IOP 4 SPB H,L; IOP 5 SPB
14	4	609-5041			VPN	Connector, Dual Row * 0.100", 50 Socket Contacts
						Manufacturer, T&B/Ansley
						Alternate: MIL-C-83503/7C-10
15	8	609-50SL			VPN	Slide Lock, D - Connectors
						Manufacturer, T&B/Ansley
16	29	PV14-4R-C			VPN	Terminal Lug, Insulated, #16-14 Wire
						Manufacturer, Panduit
17	9	PV8-56R-Q			VPN	Terminal Lug, Insulated, # 8 Wire
						Manufacturer, Panduit

F-3

F-4

SBR PRELIMINARY PARTS LIST

18	3	PV22-2R-C			VPN	Terminal Lug, Insulated, #26-22 Wire
						Manufacturer, Panduit
19	1	PV14-56R-C			VPN	Terminal Lug, Insulated, #16-14 Wire
						Manufacturer, Panduit
20	2	50-57-9403			VPN	Connector, Single Row Crimp
						Manufacturer, Molex
21	4	16-02-0103			VPN	Terminal, Female, C - Grid, #24 - 22 Wire
						Manufacturer, Molex
22	40	MS-167			VPN	Screw, Machine, 4-40 * 3/8
						Manufacturer, Spae Naur
23	40	MS-401			VPN	Washer, #4 Flat
						Manufacturer, Spae Naur
24	40	W-309			VPN	Washer, Lock, #4
						Manufacturer, Spae Naur
25	9	MS-1047			VPN	Screw, Machine, 2-56 * 3/8
						Manufacturer, Spae Naur
26	9	MW-400			VPN	Washer, Flat, #2
						Manufacturer, Spae Naur
27	9	W-307			VPN	Washer, Lock, #2
						Manufacturer, Spae Naur
28	42	382-306			VPN	Screw, Machine,
						Manufacturer, Spae Naur
29	A/R					Wire, #14 Red
30	A/R					Wire, #14 Black
31	A/R					Wire, #14 Orange
32	A/R					Wire, #14 Violet
33	A/R					Wire, #22 Red
34	A/R					Wire, #22 Black
35	A/R					Wire, #22 Yellow
36	A/R					Wire, #12 Black
37	A/R					Wire, #12 Yellow
38	10'	843-132-2801-50			VPN	Cable, Ribbon, Twist
						Manufacturer, Amphenol
39	3	28-0614			VPN	IOP Adapter Bracket
						Manufacturer, MDA
40	1	28-0615			VPN	VME P2 Adapter Bracket
						Manufacturer, MDA
41	10	4031264-07			VPN	Filler Panel
						Manufacturer, Mupac

F2 Drawings

RTSG Chassis Assembly	G442051 (3 pages)
RTSG Chassis Schematic	SDG442051 (3 pages)
ESP Chassis Assembly	G442053 (3 pages)
EST Chassis Subassembly Wiring	SDG442053 (3 pages)

THIS PAGE INTENTIONALLY LEFT BLANK

Pages F-7 to F-18

FOLD-OUT DIAGRAM G442051 sheets 1 - 3
SDG 442051 sheets 1 - 3
G 442053 sheets 1 - 3
SDG 442053 sheets 1 - 3
REMOVED PRIOR TO SCANNING -

EXTRA COPIES STORED UNDER SYSTEM NUMBER

153394

APPENDIX G

HMI DIALOGUE DESIGN

By: S. Tran

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

Page

APPENDIX G HMI DIALOGUE DESIGN G-1

G1 Utility Dialogues G-5

- G1.1 Confirmation Dialogue G-5
- G1.2 File Selection Dialogue G-6
- G1.3 Copy Dialogue G-7

G2 Satellite/Radar Definition (CRDH CSU) G-8

- G2.1 Dialogue Hierarchy Diagram G-8
- G2.2 Top Level Dialogue (crdhTopLevelD) G-9
- G2.3 Radar Concept (crdhRadarConceptD) G-13
- G2.4 Channel Selections (crdhChannelSelectionD) G-14
- G2.5 Receiver Front End (crdhRxFrontEndD) G-15
- G2.6 Multipath Definition (crdhMultipathD) G-16
- G2.7 CFA Antenna Physical (crdhCfaPhysicalD) G-17
- G2.8 Analog Effects (crdhAnalogEffectsD) G-18
- G2.9 CFA Antenna Patterns (crdhCfaAntennaPatternsD) G-19
- G2.10 CFA Antenna Operational (crdhCfaOperationalD) G-20
- G2.11 RRA Antenna Physical (crdhRraPhysicalD) G-21
- G2.12 RRA Antenna Patterns (crdhRraAntennaPatternsD) G-23
- G2.13 RRA Antenna Operational (crdhRraOperationalD) G-24
- G2.14 Auxiliary Antenna Physical (crdhAuxPhysicalD) G-25
- G2.15 Auxiliary Antenna Regular Array (crdhAuxRegularArrayD) G-26
- G2.16 Auxiliary Antenna Arbitrary Array (crdhAuxArbitraryArrayD) G-27
- G2.17 Auxiliary Antenna Operational (crdhAuxOperationalD) G-28
- G2.18 Orbital Position (crdhOrbitalPositionD) G-29
- G2.19 Data Capture Options (crdhCaptureOptionsD) G-30

G2.20 Data Reduction Options (crdhDataReductionD) G-31
G2.21 Generation Options (crdhGenerationOptionsD) G-32
G2.22 Dwell Definition (crdhDwellDefD) G-33
G2.23 NCI Definition (crdhNciDefD) G-34
G2.24 CPI Definition (crdhCpiDefD) G-35

G3 Env Definition (CEDH) CSU G-36

G4 Simulation Run Definition (CSDH CSU) G-54

G4.1 Dialogue Hierarchy Diagram G-54
G4.2 Top Level Dialogue (csdhTopLevelD) G-55
G4.3 DSP Configuration (csdhDspConfigurationD) G-57
G4.4 Formed Beam Selection (csdhFormedBeamSelD) G-58
G4.5 Blanker Selection (csdhBlankerSelD) G-65
G4.6 Nulling Systems (csdhNullingSystemsD) G-66
G4.7 Digital PMC (csdhDigitalPmcD) G-67
G4.8 Adaptive Nulling (csdhAdaptiveNullingD) G-68
G4.9 Pre-Doppler Blanking (csdhPreDopplerBlankingD) G-69
G4.10 MASR DPCA (csdhMasrDpcaD) G-70
G4.11 Pulse-Doppler (csdhPulseDopplerD) G-71
G4.12 ESP Data Capture Options (csdhEspDataCaptureD) G-72
G4.13 ESP Products (csdhEspProductsD) G-73
G4.14 Pulse Compression (csdhPulseCompressionD) G-74
G4.15 ASAR DPCA (csdhAsarDpcaD) G-75
G4.16 Range-Doppler Blanking (csdhRngDopplerBlankingD) G-76
G4.17 NCI (csdhNciD) G-77
G4.18 CFAR (csdhCfarD) G-78
G4.19 RAR (csdhRarD) G-79
G4.20 RSP Data Capture Options (csdhRspDataCaptureD) G-80
G4.21 RSP Products (csdhRspProducts) G-81
G4.22 Data Reduction Options (csdhDataReductionD) G-82
G4.23 Dwell Level Pulse Doppler (csdhDiPulseDopplerD) G-83
G4.24 Dwell Level ASAR DPCA (csdhDiAsarDpcaD) G-84
G4.25 Dwell Level CFAR (csdhDiCfarD) G-85
G4.26 Dwell Level RAR (csdhDiRarD) G-86
G4.27 NCI Level Digital PMC (csdhNciDigitalPmc) G-87
G4.28 NCI Level NCI (csdhNciNciD) G-88

G5 Monitor/Control Signal Generation (CGEH CSU) G-89

G5.1 Top Level Dialogue (cgehTopLevelD) G-89
G5.2 Progress Status Dialogue G-91

G6 Monitor/Control Simulation Run (CSIH CSU) G-92

G6.1 Top Level Dialogue (csihTopLevelD) G-92
G6.2 Progress Status Dialogue G-94

APPENDIX G

HMI DIALOGUE DESIGN

This appendix contains the layout design and descriptions for the HMI dialogues that are used in the following tasks:

- Satellite/Radar Definition task (CRDH CSU)
- Environment Definition task (CEDH CSU)
- Simulation Run Definition task (CSDH CSU)
- Monitor/Control Signal Generation task (CGEH CSU)
- Monitor/Control Simulation Run task (CSIH CSU)

This appendix provides information helpful in reviewing the implementation of the user interface applications provided by the SPD Testbed Controller. In addition, it captures the implementation information which is not explicitly known by the HMI parameter tables. The parameters required by the tasks mentioned above are described in the Satellite and Radar HMI parameter table, Environment HMI parameter table and Simulation Run HMI parameter table in Appendix A of the Testbed Specification (Document A-2). The approach and standards to be used in designing the HMI dialogues are described in Appendix A of the Testbed Preliminary Design document (Document R-1). The OSF/Motif Widget set is used in building the HMI dialogues. For more information on Motif style and widget behaviour, please refer to the documentation from the Open Software Foundation (Documents R-14, R-15, R-16 and R-17).

Conventions

1. All dialogue boxes have the following standard push buttons that typically complete the editing operation:

• OK	apply all changes made in this dialogue box to in-memory data structure, close and exit this window.
• Cancel	do not apply changes made in this dialogue box to in-memory data structure, close and exit this window.

- Apply apply the changes made in this dialogue box to in-memory data structure, and do not exit this window. This button is optional for a few dialogues.

2. Customized latitude/longitude widget which provides input of latitude, longitude, azimuth angle, or attitude angles in degrees, minutes and seconds (ddd:mm:ss) format.

Range of latitude:	-90 (South) to +90 (North)
Range of longitude:	-180 (West) to +180 (East)

- Clicking page increment/decrement region or clicking line increment/decrement button (arrow) increases or decreases minutes or seconds, respectively.

3. Customized scale widget contains a label widget, a slider and a text field widget. The label widget displays the parameter name and units. In response to the slider movement, the text field displays the relative slider value in integer or floating point value. In addition, the text field accepts direct textual input to extend resolution which is normally restricted by the slider input; for example, the user can add more digits after the default decimal place.

4. Channel acronym naming convention used to describe a channel in the channel selection dialogues (See Blanker Selection dialogue G4.5, Nulling System dialogue G4.6) are as follows:

Main antenna/Phase centres	P1 (RRA: Main antenna) P1 P2 P3 (group 1) P1 P2 P3 (group 2): CFA
Auxiliary	A1 A2 A3 ... A16
Delay tap	T1 T2 T3

For example, all channels of a CFA antenna are illustrated in the following table:

DPCA Group 1	DPCA Group 2	Auxiliary #1	Auxiliary #2	Auxiliary #3	Auxiliary #4
P1	P1	A1	A2	A3	A4
P2	P2	T1	T1		T1
		T2			T2
		T3			

The description for the channel assignment above appears in the dialogue window as follows:

channel acronym:	P1	P2	P1	P2	A1	T1	T2	T3	A2	T1	A3	A4	T1	T2
channel ID:	1	2	3	4	5	6	7	8	9	10	11	12	13	14

The channel ID of the main beam, phase centre, auxiliary and delay tap are displayed with the associated channel acronym. The channel ID uniquely identifies a channel according to the channel mapping algorithm described in the Dependency logic for Simulation Run HMI in Appendix A of the Testbed Specification (Document A-2).

5. A grid button area (arrays of buttons) is sometimes included in an appropriate dialogue to allow the user to select one or more channels. Toggling the buttons on the grid button area indicates which channels are selected:

   - "on" (pressed, filled):selected;
   - "off" (unpressed, empty):unselected.

6. The user can specify filenames in dialogue boxes by pressing the small button to the left of the desired file name text field. A file selection dialogue box will appear to allow the user to choose a file. The user is not permitted to enter a filename directly into the filename field; the field is for display only.

Implementation Rules

1. Structural parameter dependency is implemented in the dialogue design via ghosting of certain widgets.

2. Most of the parameter value dependent ranges (referred to as smart ranges) are implemented as absolute (fixed) ranges with the exception of those which are noted in this appendix.

3. Most of the parameter value dependent defaults (referred to as smart defaults) are implemented as absolute (fixed) defaults with the exception of those which are dependent only on the Concept parameter.

4. In a text field input, if a numeric entry is out of range, it is set to the nearest range limit. If invalid input is detected (e.g. entering non-numeric characters), the previous numeric value will be retained.

The HMI dialogue layouts and descriptions are organized as follows:

- G1 Utility Dialogues
  - G1.1 Confirmation Dialogue
  - G1.2 File Selection Dialogue
  - G1.3 Copy Dialogue
- G2 Satellite/Radar Definition (CRDH)
  - G2.1 Dialogue Hierarchy Diagram
  - G2.2 Top Level Dialogue (crdhTopLevelD)
  - G2.3 Radar Concept (crdhRadarConceptD)
  - G2.4 Channel Selections (crdhChannelSelectionsD)
  - G2.5 Receive Front End (crdhRxFrontEndD)
  - G2.6 Multipath Definition (crdhMultipathD)
  - G2.7 CFA Antenna Physical (crdhCfaPhysicalD)
  - G2.8 Analog Effects (crdhAnalogEffectsD)
  - G2.9 CFA Antenna Patterns (CrdhCfaAntennaPatternsD)
  - G2.10 CFA Antenna Operational (crdhCFAOperaionalD)
  - G2.11 RRA Antenna Physical (crdhRraPhysicalD)
  - G2.12 RRA Antenna Patterns (crdhRraAntennaPatternsD)
  - G2.13 RRA Antenna Operational (crdhRraOperationalD)
  - G2.14 Auxiliary Antenna Physical (crdhAuxPhysicalD)

G2.15 Auxiliary Antenna Regular Array (crdhAuxRegular ArrayD)
G2.16 Auxiliary Antenna Arbitrary Array (crdhAuxArbitraryArrayD)
G2.17 Auxiliary Antenna Operational (crdhAuxOperationalD)
G2.18 Orbital Position (crdhOrbitalPositionD)
G2.19 Data Capture Options (crdhCaptureOptionsD)
G2.20 Data Reduction Options (crdhDataReductionD)
G2.21 Generation Options (crdhGenerationOptionsD)
G2.22 Dwell Definition (crdhDwellDefD)
G2.23 NCI Definition (crdhNcidefD)
G2.24 CPI Definition (crdhCpiDefD)

G3 Environment Definition (CEDH)

G4 Simulation Run Definition (CSDH)
G4.1 Dialogue Hierarchy Diagram
G4.2 Top Level Dialogue (csdhTopLevelD)
G4.3 DSP Configuration (csdhDspConfigurationD)
G4.4 Formed Beam Selection (csdhFormedBeamSelD)
G4.5 Blanker Selection (csdhBlankerSelD)
G4.6 Nulling Systems (csdhNullingSystems)
G4.7 Digital PMC (csdhDigitalPmcD)
G4.8 Adaptive Nulling (csdhAdaptiveNullingD)
G4.9 Pre-Doppler Blanker (csdhPreDopplerBlankerD)
G4.10 MASR DPCA (csdhMasrDpcaD)
G4.11 Pulse-Doppler (csdhPulseDopplerD)
G4.12 ESP Data Capture Options (csdhEspDataCaptureD)
G4.13 ESP Products (csdhEspProductsD)
G4.14 Pulse Compression (csdhPulseCompressionD)
G4.15 ASAR DPCA (csdhAsarDpcaD)
G4.16 Range-Doppler Blanking (csdhRngDopplerBlankingD)
G4.17 NCI (csdhNciD)
G4.18 CFAR (csdhCfarD)
G4.19 RAR (csdhRarD)
G4.20 RSP Data Capture Options (csdhRspDataCaptureD)
G4.21 RSP Products (csdhRspProducts)
G4.22 Data Reduction Options (csdhDataReductionD)
G4.23 Dwell Level Pulse Doppler (csdhDiPulseDopplerD)
G4.24 Dwell Level ASAR DPCA (csdhDiAsarDpcaD)
G4.25 Dwell Level CFAR (csdhDiCfarD)
G4.26 Dwell Level RAR (csdhDiRarD)
G4.27 NCI Level Digital PMC (csdhNciDigitalPmc)
G4.28 NCI Level NCI (csdhNciNciD)

G5 Monitor/Control Signal Generation (CGEH)
G5.1 Top Level Dialogue (cgehTopLevelD)
G5.2 Progress Status Dialogue

G6 Monitor/Control Simulation Run (CSIH)
G6.1 Top Level Dialogue (csihTopLevelD)
G6.2 Progress Status Dialogue

G1 UTILITY DIALOGUES

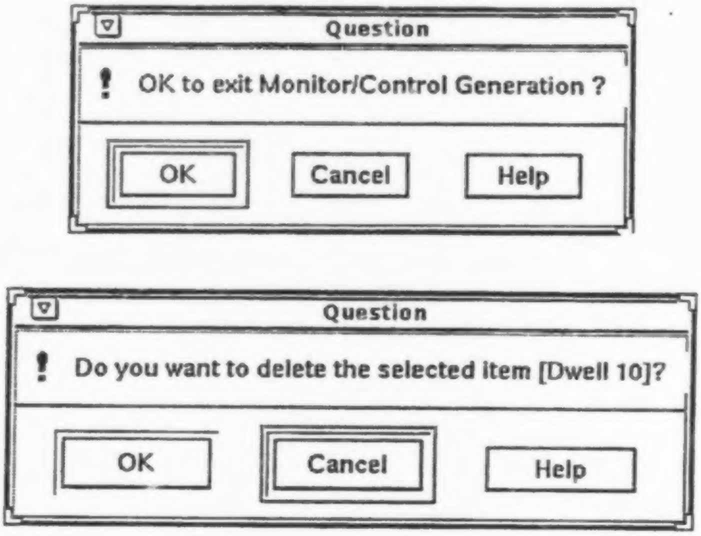
These are the common dialogue boxes which are used in various HMI applications of the Testbed Controller.

G1.1 Confirmation Dialogue

The confirmation dialogue is displayed for getting user input for confirmation to proceed with an operation. The callback function performs "synchronous" input from the user. i..e. it enters an internal event loop until the user clicks "OK" or "Cancel". It appears to block while waiting for input. It is used in the following cases:

- exiting an application;
- changing radar concept;
- deleting sub-intervals (dwells, NCIs, CPIs);
- proceeding with Signal Generation or Simulation Run with erroneous parameters which have been detected by CVAL black-box validation;
- reporting warning or error messages.

When the "help" button is pressed, a text description window containing information about the confirmation dialogue is displayed.



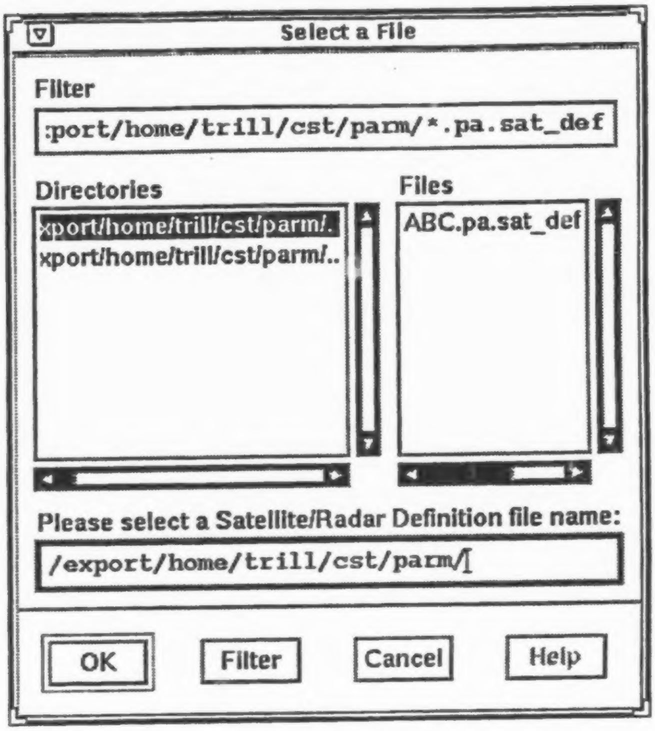
! OK to exit Monitor/Control Generation ?

! Do you want to delete the selected item [Dwell 10]?

Confirmation_Dialogue

G1.2 File Selection Dialogue

This dialogue box provides a means to select a file or a directory by traversing through directories and viewing the files and sub-directories in them. A text input field is provided for displaying and editing a directory mask to select the files to be displayed. The directory mask can contain one or more wildcard characters in its directory portion, its file portion, or both. All files or directories which are matched with the directory mask will be displayed, when the user activates the "Filter" button. Another text input field is provided for displaying and editing a filename or a directory name. The user can type in directly a known filename or a known directory name.



Filter
:port/home/trill/cst/parm/*.pa.sat_def

Directories
xport/home/trill/cst/parm/.
xport/home/trill/cst/parm/..

Files
ABC.pa.sat_def

Please select a Satellite/Radar Definition file name:
/export/home/trill/cst/parm/

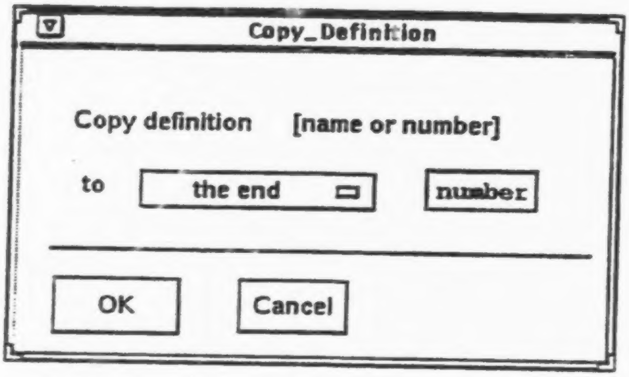
cgehSelectFile

G1.3 Copy Dialogue

This dialogue is used in copying an interval definition (Copy Dwell, Copy NCI, Copy CPI). It allows the user to enter the destination: the user can either select "before position", enter a valid interval identifier (dwell id, NCI id or CPI id) to add a new interval *before* the destination or select "the end" or "the beginning" button to add a new interval at the end or the beginning of intervals. For example, the current dwell is 5 of a total of 10 dwells. Entering "9" in the "number" text input field copies dwell number 5 to dwell number 9, after dwell number 9 and 10 were re-named to dwell 10 and 11 respectively. Pressing "the end" button creates dwell number 11 with dwell number 5.

The default selection is "the end". In this case, the "number" field is for display only and a number corresponding to the last interval defined is displayed. For example, "11" is displayed if 10 dwells have been defined before entering the copy dialogue.

When "the beginning" is selected, a "1" is displayed automatically in the "number" field and this field is not user modifiable. When "before position" is selected, a number corresponding to the last interval defined is provided as a default. An out of range number entered is rounded to the nearest range limit.



Copy_Definition

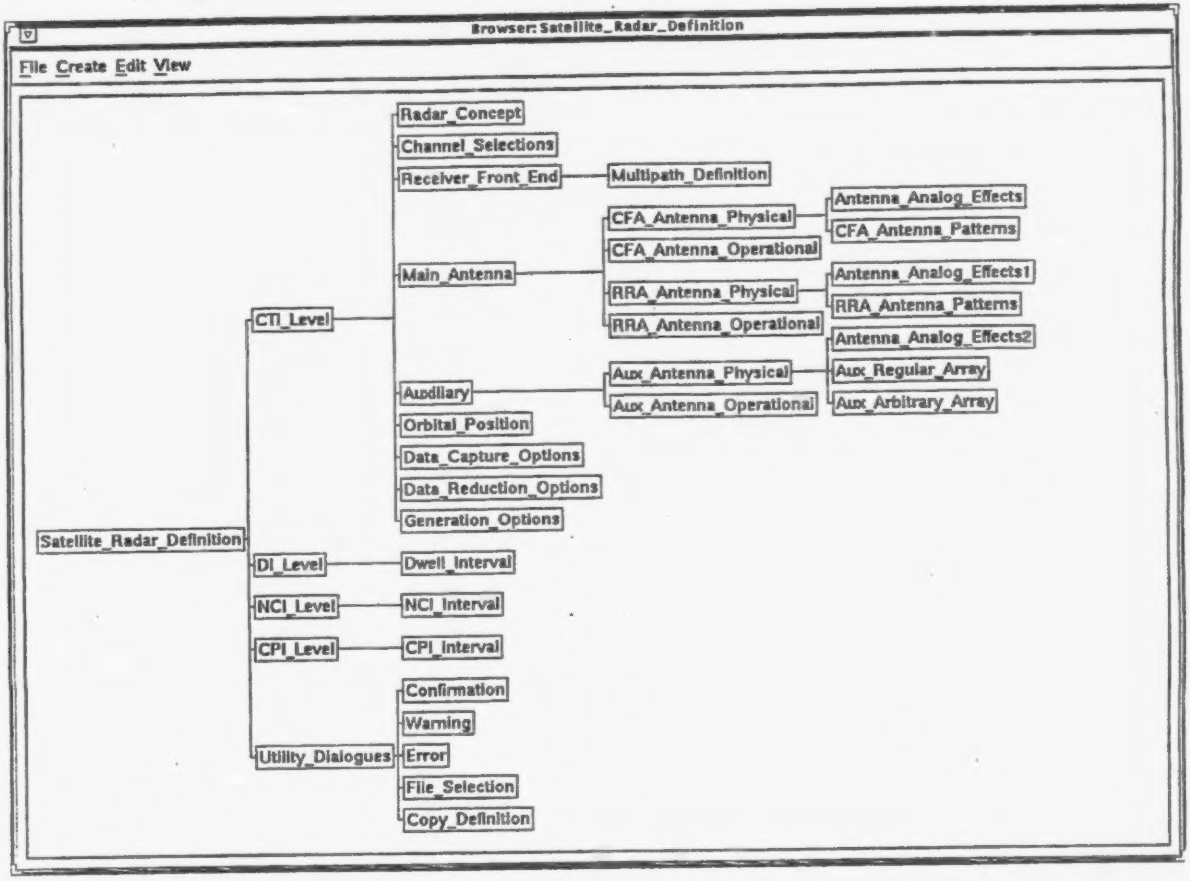
Copy definition [name or number]

to the end number

G2 SATELLITE/RADAR DEFINITION (CRDH CSU)

G2.1 Dialogue Hierarchy Diagram

The Satellite/Radar Definition HMI task is implemented by utilizing a tree-like structure of dialogue boxes. Each dialogue box represents a parameter group in the definition. The structure of the dialogue boxes is designed based on the Satellite/Radar Definition organization described in the Satellite and Radar HMI parameter table. The dialogue hierarchy can be depicted in the following hierarchy diagram.


Browser: Satellite_Radar_Definition
File Create Edit View
Satellite_Radar_Definition
CTI_Level
Radar_Concept
Channel_Selections
Receiver_Front_End
Multipath_Definition
Main_Antenna
CFA_Antenna_Physical
Antenna_Analog_Effects
CFA_Antenna_Patterns
CFA_Antenna_Operational
RRA_Antenna_Physical
Antenna_Analog_Effects1
RRA_Antenna_Patterns
RRA_Antenna_Operational
Auxiliary
Aux_Antenna_Physical
Antenna_Analog_Effects2
Aux_Regular_Array
Aux_Arbitrary_Array
Aux_Antenna_Operational
Orbital_Position
Data_Capture_Options
Data_Reduction_Options
Generation_Options
DI_Level
Dwell_Interval
NCI_Level
NCI_Interval
CPI_Level
CPI_Interval
Utility_Dialogues
Confirmation
Warning
Error
File_Selection
Copy_Definition

Satellite_Radar_Definition

G2.2 Top Level Dialogue (crdhTopLevelD)

This is the top level dialogue box of the Satellite/Radar Definition HMI task. It is designed as a "visualization" window to provide the outlook of the structure of the Satellite/Radar Definition. Please see Section A3, Visualization Window Design in Appendix A of the Testbed Preliminary Design document (Document R-1) for more details of the design concept of this dialogue.

The top level dialogue has six main areas:

- A pull-down menu bar and a text field to display the Satellite/Radar Definition filename
- A CTI Level definition area
- A Dwell Level definition area
- A NCI Level definition area
- A CPI Level definition area
- A Validation Error Log area

Each area is designed as a paned window. The user can adjust the size of an area by scrolling the sash up or down; the sash appears as a square box positioned on the bottom of the area. A typical usage of the paned window is to enlarge Validation Error Log area to allow more messages to be displayed.

The key feature of the visualization window is that it gives the user a quick look at both "functional" and "temporal" associations. The parameters are grouped by function and each parameter group is represented as a push button on the visualization window. Clicking one of these buttons invokes the "Edit" function. Depending on the changeability level, the parameter group buttons appear in either the CTI, Dwell, NCI or CPI Level area. The user can select a dwell, a NCI or a CPI interval via the scrolled button list in the selection sub-area of the associated level area. The selection display indicates which dwell, NCI and CPI are current. When the user clicks on a parameter group button, a lower-level dialogue box is popped-up for entering or changing parameters associated with the selected parameter group. The sub-intervals are organized as a tree-like structure which is implemented using linked lists corresponding to interval levels. It provides fast access to a sub-interval for both modification and deletion functions. The linked lists are converted to the structure defined in CRAD CSU before writing the Satellite/Radar Definition file. Similarly, the structure is converted to the linked list after reading.

The constraints in the HMI operation are:

- Only one sub-interval can be selected to be current at a temporal level;
- Only one parameter group can be activated at a time; i.e. the user must close the current window first before he can choose another parameter group;
- In the case where a dialogue box has lower-level dialogues currently opened, this dialogue can not be closed until all lower-level dialogues have been closed.

Except for the "Editing" function mentioned above, a function is typically activated by choosing a menu in the pull-down menu bar located on the top of this dialogue. The pull-down menu is described in a Backup Naur like format as follows:

```
pull-down menu
{                                   Actions
    File
    {
```

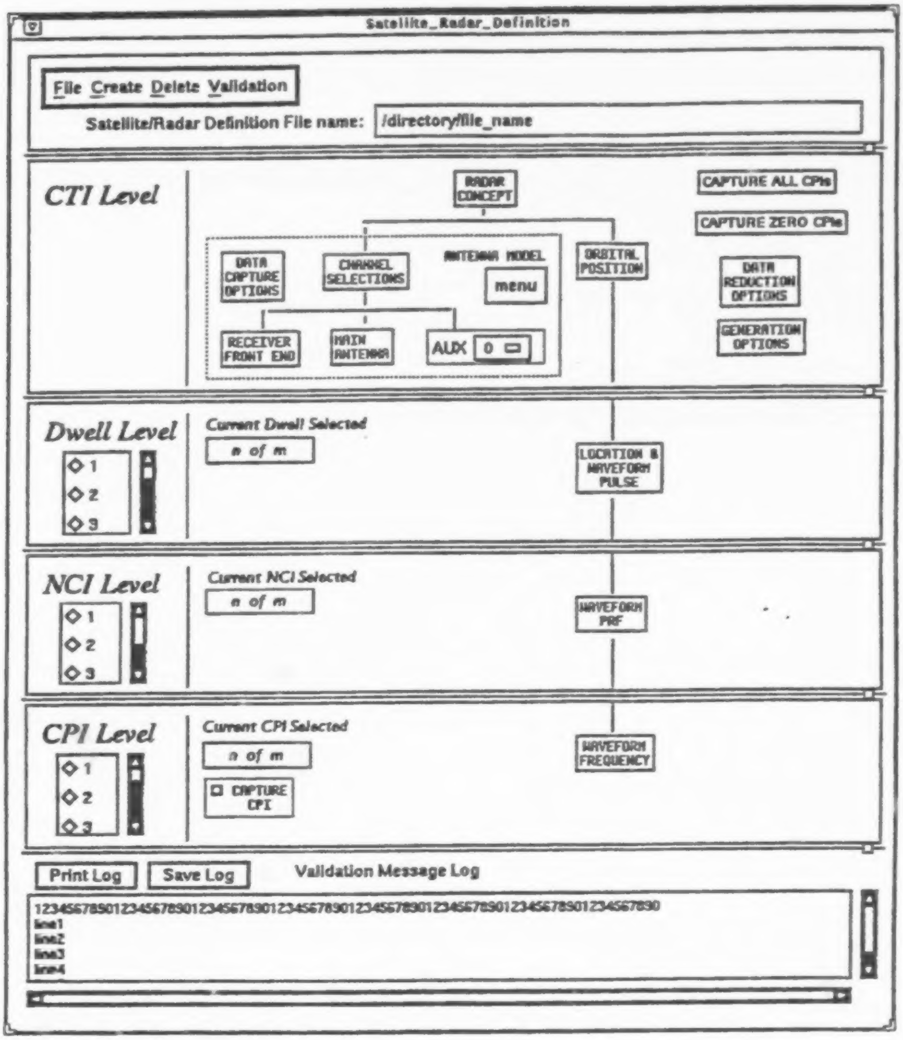
New	Create an in-memory definition structure with defaults. The definition name is defaulted to "Untitled". The user can then use the "Save" function to save the new definition to the named file.
Open	Open an existing Satellite/Radar Definition file. A file selection dialogue appears for selecting an existing Satellite/Radar Definition file to be read into an in-memory definition structure.
Save	Save the in-memory definition structure to file.
Save As	Save the in-memory definition structure to the named file. A file selection dialogue appears for entering file name.
Exit	Prompt the user with the confirmation dialogue. If the user clicks "OK", pop-down all sub-level windows if any, close and exit this window. A warning dialogue will be displayed, if the user made changes and has not yet performed a "Save" function.
}	
Create	
{	
New	
{	
Dwell	Create a new dwell. The new dwell is set with defaults and positioned at the end of the dwell linked list.
NCI	Create a new NCI interval in the current dwell. The new NCI interval is set with defaults and positioned at the end of the NCI linked list.
CPI	Create a new CPI interval in the current dwell and NCI interval. The new CPI is set with defaults and positioned at the end of the CPI linked list.
}	
Copy	
{	
Dwell	Create a new dwell using the current dwell. A copy dialogue appears for specifying the dwell destination. The new dwell is inserted or appended to the dwell linked list.
NCI	Create a new NCI interval with the current NCI interval parameters. A copy dialogue appears for specifying the NCI destination. The new NCI interval is inserted or appended to the NCI linked list.
CPI	Create a new CPI interval with the current CPI interval parameters. A copy dialogue appears for specifying the CPI destination. The new CPI interval is inserted or appended to the CPI linked list.
}	
}	
Delete	
{	
Dwell	Delete the current dwell. A confirmation dialogue is displayed for confirmation. If "yes" button is clicked, the current dwell is removed from the dwell linked list.
NCI	Delete the current NCI. A confirmation dialogue is displayed for confirmation.. If "yes" button is clicked, the current NCI interval is removed from the NCI linked list.
CPI	Delete the current CPI. A confirmation dialogue is displayed for confirmation. If "yes" button is clicked, the current CPI interval is removed from the CPI linked list.
}	
Validate	
{	

```
        start          Start a black-box validation using CVAL. The error messages
                       returned from the CVAL function will be displayed in the Validation
                       Error Log area.
    }
}
```

At the beginning of Satellite/Radar definition, the user selects a radar concept via the Radar Concept dialogue input. The system will provide default values to parameters associated with the antenna definition. The change of concept attribute from CFA to RRA or from RRA to CFA affects all parameters which have structural dependency on the concept type, such as main antenna structure, and auxiliary structure including analog effects and antenna patterns. In addition, smart defaults and smart range check will be implemented for the following parameters which have dependency on concept: frequency band, auxiliary type, site (rotating/non-rotating), beam centre location, pulse duration, QPM, and peak pulse power.

The Antenna Model associates with the following parameter groups: Radar Concept, Data Capture Options, Channel Selections, Receiver-Front End, Main Antenna and Auxiliaries. The HMI supports re-usable Antenna Models, including functions to read antenna model parameters from file and saving back to file. A pull-down menu with "New", "Open", "Save", "Save As" functions can be activated by clicking the menu selection button in the Antenna Model sub-area. A file selection dialogue appears for selecting an existing antenna model file when "Open" function is chosen. The antenna type (RRA or CFA) of the selected Antenna Model file will be matched with the concept attribute defined in the Radar Concept parameter group.

When pressing the "Main Antenna" and "AUX" button, a pop-up menu will be displayed to allow the user to pick either a "Physical" or "Operational" characteristics definition. Since there are multiple instances of auxiliaries, the user must select an auxiliary before editing auxiliary characteristics. This can be done by selecting a button in the option menu located beside the "AUX" button.


Satellite_Radar_Definition
File Create Delete Validation
CTI Level
RADAR CONCEPT
DATA CAPTURE OPTIONS
CHANNEL SELECTIONS
ANTENNA MODEL
menu
ORBITAL POSITION
RECEIVER FRONT END
MAIN ANTENNA
AUX 0
CAPTURE ALL CPIs
CAPTURE ZERO CPIs
DATA REDUCTION OPTIONS
GENERATION OPTIONS
Dwell Level
◇1
◇2
◇3
LOCATION & WAVEFORM PULSE
NCI Level
◇1
◇2
◇3
WAVEFORM PRF
CPI Level
◇1
◇2
◇3
☐ CAPTURE CPI
WAVEFORM FREQUENCY
Print Log
Save Log
Validation Message Log

Satellite/Radar Definition File name: /directory/file_name

Current Dwell Selected: n of m

Current NCI Selected: n of m

Current CPI Selected: n of m

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
line1
line2
line3
line4
```

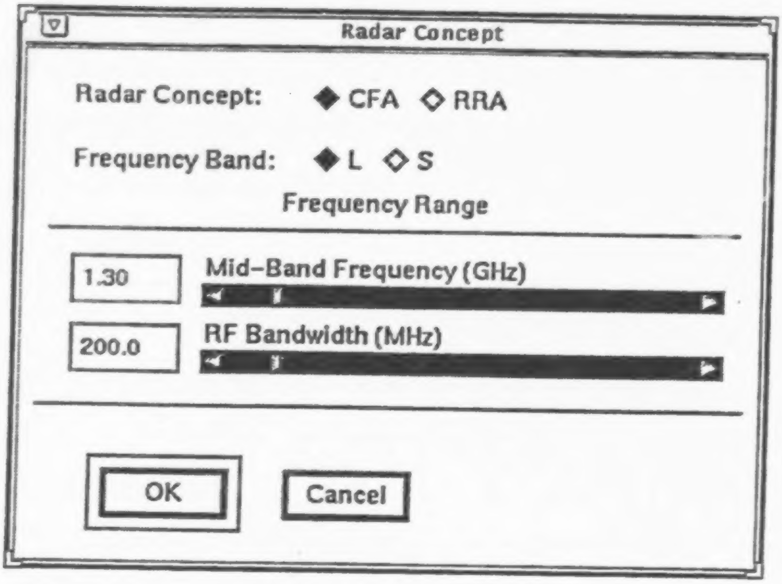
crdhTopLevelD

G2.3 Radar Concept (crdhRadarConceptD)

This dialogue box permits the user to make changes to parameters associated with the "Radar Concept" definition described in Satellite/Radar HMI parameter Table 1.1.

Fixed ranges will be implemented for the following parameters:
1 <= mid-band frequency <= 4 GHz

Smart defaults will be implemented for the following parameters:
- frequency band
- mid-band frequency
- RF bandwidth



Radar Concept

Radar Concept: ◆ CFA ◇ RRA

Frequency Band: ◆ L ◇ S

Frequency Range

Mid-Band Frequency (GHz): 1.30

RF Bandwidth (MHz): 200.0

crdhRadarConceptD

G2.4 Channel Selections (crdhChannelSelectionsD)

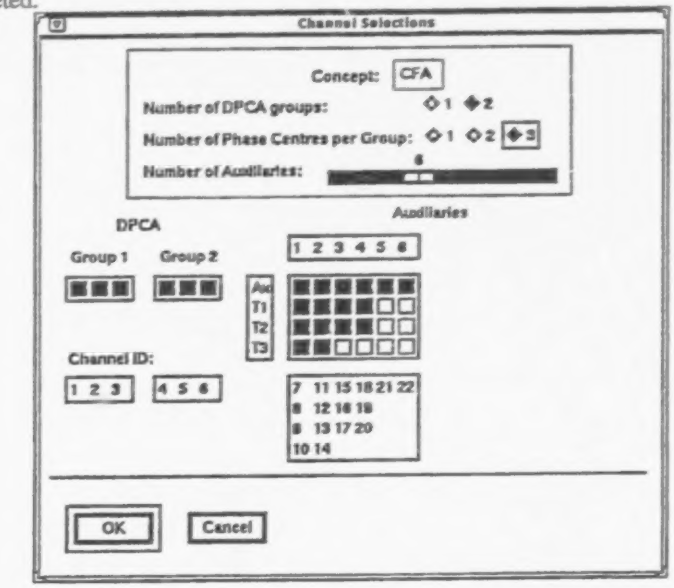
This dialogue box permits the user to make changes to parameters associated with antenna channel selections (RRA or CFA) described in Satellite/Radar HMI parameter Table 2.2.6, 2.2.6.1, 2.4, 2.4.5.

The user first enters the number of DPCA groups, the number of phase centres per group and the number of auxiliaries in the textual input area. Concept text field is display only. For RRA radar concept, both the number of DPCA groups and the number of phase centres per group is set to 1; i.e. no input is allowed. Push buttons associated with the available phase centres and auxiliaries (first row of push buttons) will be displayed as "on" (pressed or filled) and are insensitive.

After the numbers of phase centres and auxiliaries have been defined, the user can graphically select or unselect delay taps by toggling push buttons on the grid button area. The button state indicates whether the delay tap is selected: "on" (pressed or filled) indicating "selected" or "off" (unpressed) indicating "unselected". The user presses buttons to define delay taps (up to 3) for each available auxiliary antenna. Validation is executed to ensure that the total number of auxiliaries and delay taps does not exceed 16. Delay taps for an auxiliary may be unselected starting from the highest numbered tap.

A channel refers to main beam, phase centre, auxiliary or delay tap. Channel IDs are derived using the channel mapping algorithm described in the Dependency Logic for Simulation Run HMI of the Testbed Specification (Document A-2). The Channel IDs are always displayed and are updated every time the user changes the channel selections mentioned above.

The user can change the number of auxiliaries as long as the total number of auxiliaries and taps does not exceed 16. If number of auxiliaries is reduced, the highest numbered auxiliaries are deleted.



Concept: CFA

Number of DPCA groups: ◇ 1 ◆ 2

Number of Phase Centres per Group: ◇ 1 ◇ 2 ◆ 3

Number of Auxiliaries: 6

Channel ID:

1 2 3

4 5 6

7 11 15 18 21 22
8 12 16 19
9 13 17 20
10 14

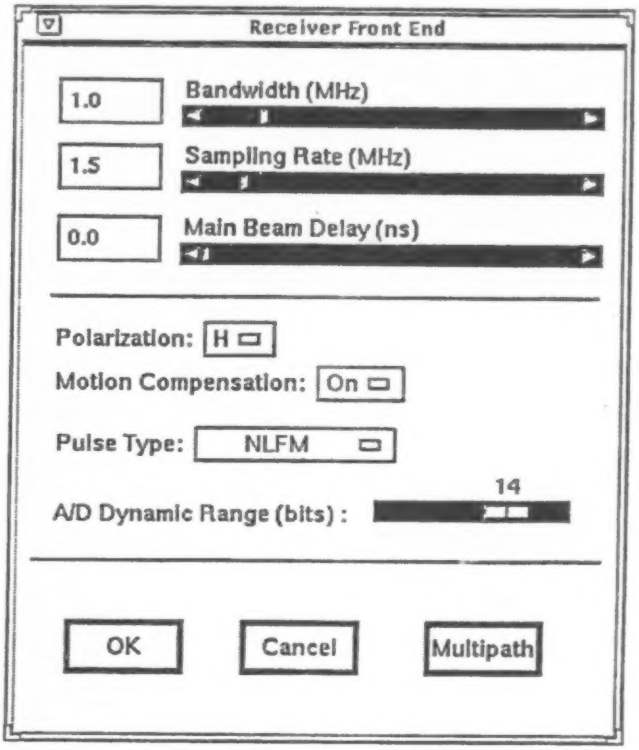
crdhChannelSelectionsD

G2.5 Receiver Front End (crdhRxFrontEndD)

This dialogue box permits the user to make changes to parameters associated with the "Receiver Front End" definition described in Satellite/Radar HMI parameter Table 2.5.

Pressing the "Multipath" button pops up a Multipath Definition dialogue (See Section G2.6) for editing multipath definition.

Fixed defaults is implemented for the following parameters:
- main beam delay: 0 ns.



Receiver Front End

1.0 Bandwidth (MHz)

1.5 Sampling Rate (MHz)

0.0 Main Beam Delay (ns)

Polarization: H

Motion Compensation: On

Pulse Type: NLFM

A/D Dynamic Range (bits) : 14

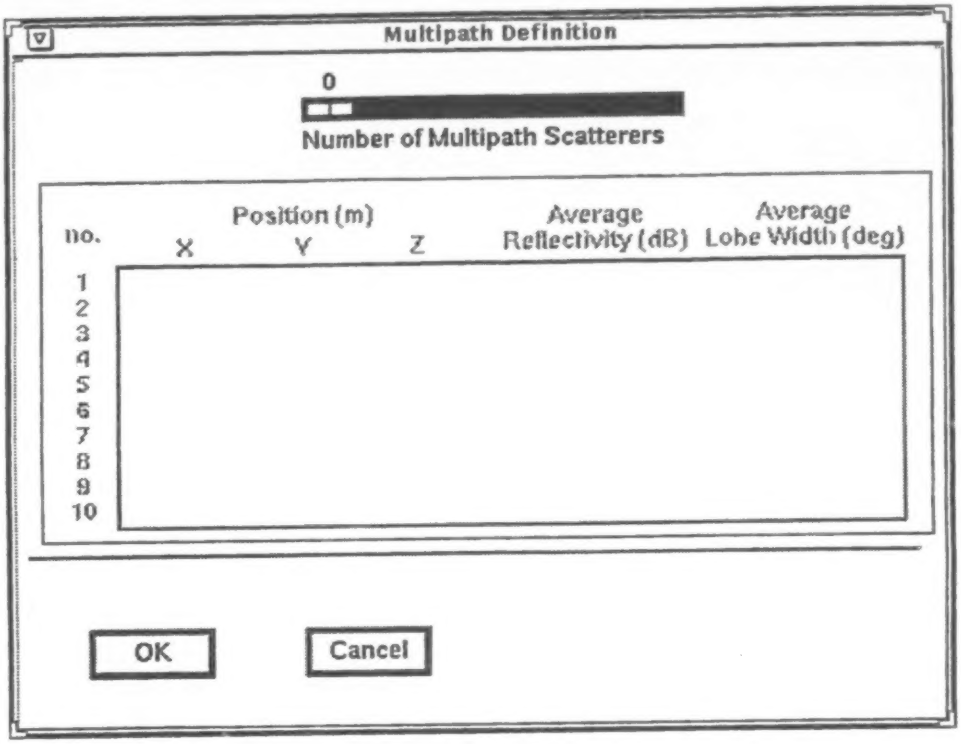
crdhRxFrontEndD

G2.6 Multipath Definition (crdhMultipathD)

This dialogue box permits the user to make changes to parameters associated with the "Multipath" definition described in Satellite/Radar HMI parameter Table 2.6.

For a new Satellite/Radar definition, the number of multipath scatterers is initialized to zero, the entire input area is ghosted and data input is disabled. Increasing the number of multipath scatterers enables parameter input.

The selected multipath scatterers are numbered from 1 up to the total number of scatterers. Only the selected multipath scatterers are sensitive; the unselected ones are ghosted.


Multipath Definition
0
Number of Multipath Scatterers
no.
Position (m)
X
Y
Z
Average Reflectivity (dB)
Average Lobe Width (deg)
1
2
3
4
5
6
7
8
9
10
OK
Cancel

crdhMultipathD

G2.7 CFA Antenna Physical (crdhCfaPhysicalD)

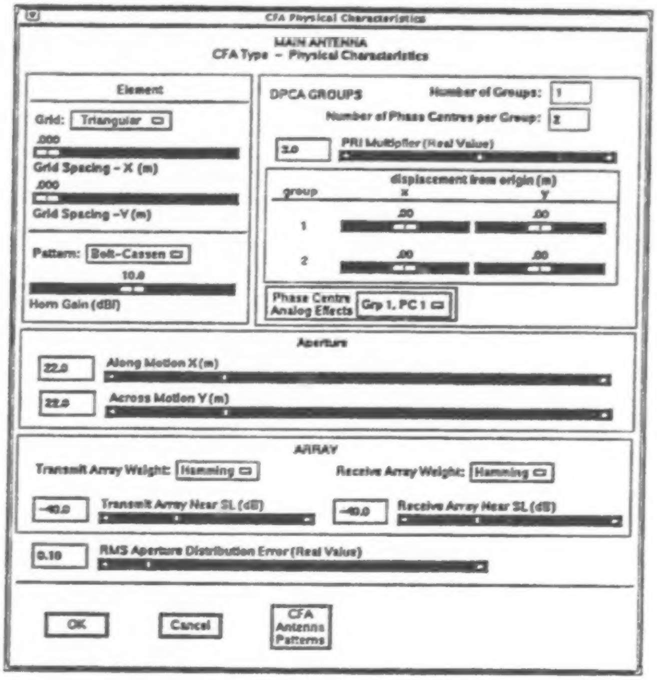
This dialogue box permits the user to make changes to parameters associated with the "CFA Antenna Physical" characteristics definition described in Satellite/Radar HMI parameter Table 2.2 excluding 2.2.5, 2.2.6 and 2.2.6.1.

The number of groups and the number of phase centres per group is not changeable in this dialogue. They can only be modified in the "Channel Selections" dialogue. The unselected phase centres input is disabled by ghosting certain widgets in the "DPCA groups" area. Choosing a phase centre is required before invoking the edit function for phase centre analog effects. The user chooses a phase centre by selecting a button in the option menu located beside the "Phase Centre Analog Effects" button. Pressing a "Phase Centre Analogue Effects" button activates the analog effects definition (See Section G2.8).

Clicking the "CFA Antenna Patterns" push button pops up the CFA Antenna Patterns dialogue (See Section G2.9) for editing the antenna patterns definition.

Fixed ranges will be implemented for the following parameters:
0 <= grid spacing (dX and dY) <= 0.3 m
-50 <= displacement from aperture centre (X and Y) <= 50 m

Fixed defaults will be implemented for the following parameters:
grid spacing (dX and dY) 0.115 m



CFA Physical Characteristics

MAIN ANTENNA
CFA Type – Physical Characteristics

Element

Grid: Triangular

Grid Spacing – X (m): .000

Grid Spacing – Y (m): .000

Pattern: Bolt-Cassen

Horn Gain (dBI): 10.0

DPCA GROUPS

Number of Groups: 1

Number of Phase Centres per Group: 2

PRI Multiplier (Real Value): 3.0

group	displacement from origin (m) x	y
1	.00	.00
2	.00	.00

Phase Centre Analog Effects: Grp 1, PC 1

Aperture

Along Motion X (m): 22.0

Across Motion Y (m): 22.0

ARRAY

Transmit Array Weight: Hamming

Receive Array Weight: Hamming

Transmit Array Near SL (dB): -40.0

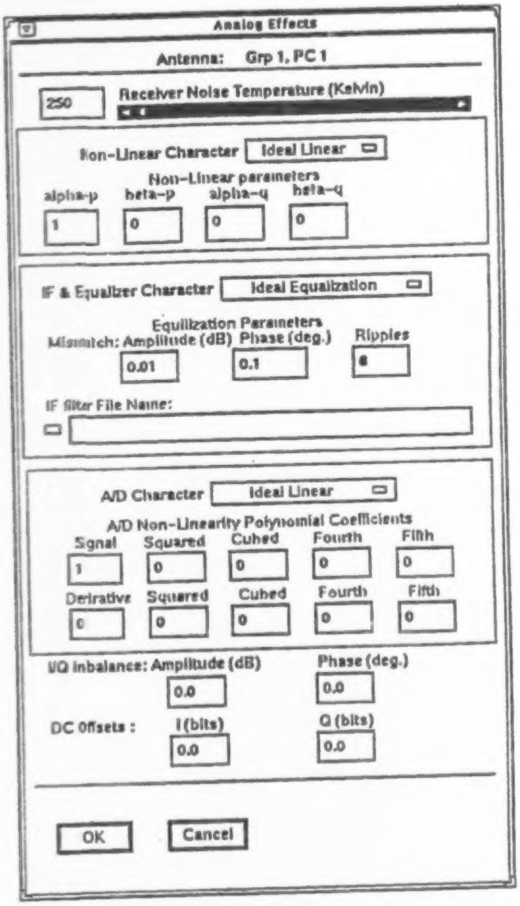
Receive Array Near SL (dB): -40.0

RMS Aperture Distribution Error (Real Value): 0.10

crdhCfaPhysicalD

G2.8 Analog Effects (crdhAnalogEffectsD)

This dialogue box permits the user to make changes to parameters associated with the "Analog Effects" definition described in Satellite/Radar HMI parameter Table 2.5.5.



Analog Effects

Antenna: Grp 1, PC 1

250 Receiver Noise Temperature (Kelvin)

Non-Linear Character: Ideal Linear

Non-Linear parameters

alpha-p	beta-p	alpha-q	beta-q
1	0	0	0

IF & Equalizer Character: Ideal Equalization

Equilization Parameters

Mismatch: Amplitude (dB)	Phase (deg.)	Ripples
0.01	0.1	6

IF filter File Name:

A/D Character: Ideal Linear

A/D Non-Linearity Polynomial Coefficients

Signal	Squared	Cubed	Fourth	Fifth
1	0	0	0	0

Derivative	Squared	Cubed	Fourth	Fifth
0	0	0	0	0

I/Q Inbalance: Amplitude (dB)	Phase (deg.)
0.0	0.0

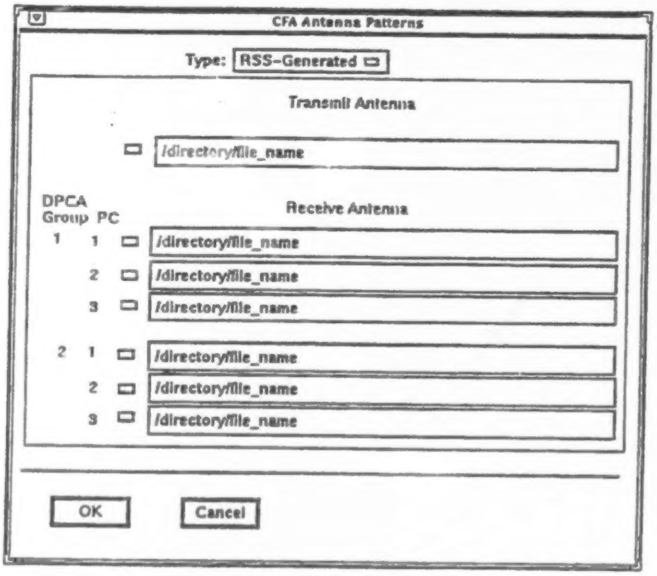
DC Offsets: I (bits)	Q (bits)
0.0	0.0

crdhAnalogEffectsD

G2.9 CFA Antenna Patterns (crdhCfaAntennaPatternsD)

- This dialogue box permits the user to make changes to parameters associated with the "CFA Antenna Patterns" definition described in Satellite/Radar HMI parameter Table 2.2.10, 2.2.10.1.

If "RSS-Generated" is selected, the input area is dimmed to disable parameter input.



Type: RSS-Generated

Transmit Antenna

/directory/file_name

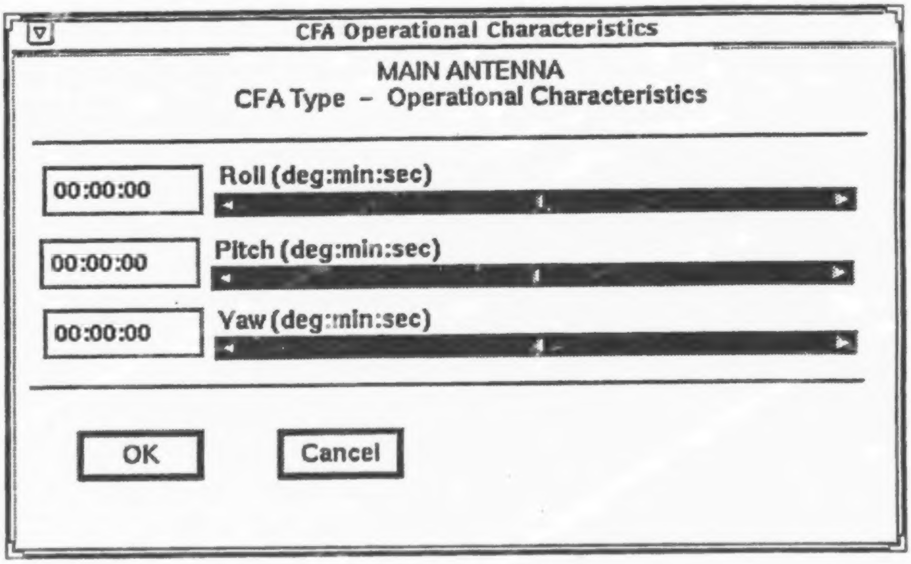
Receive Antenna

DPCA Group	PC	
1	1	/directory/file_name
	2	/directory/file_name
	3	/directory/file_name
2	1	/directory/file_name
	2	/directory/file_name
	3	/directory/file_name

crdhCfaAntennaPatternD

G2.10 CFA Antenna Operational (crdhCfaOperationalD)

This dialogue box permits the user to make changes to parameters associated with the "CFA Antenna Operational" characteristics definition described in Satellite/Radar HMI parameter Table 2.2.5.



CFA Operational Characteristics

MAIN ANTENNA
CFA Type – Operational Characteristics

Roll (deg:min:sec): 00:00:00
Pitch (deg:min:sec): 00:00:00
Yaw (deg:min:sec): 00:00:00

crdhCfaOperationalD

G2.11 RRA Antenna Physical (crdhRraPhysicalD)

This dialogue box permits the user to make changes to parameters associated with the "RRA Antenna Physical" characteristics definition described in Satellite/Radar HMI parameter Table 2.3 excluding 2.3.13 to 2.3.17.1.

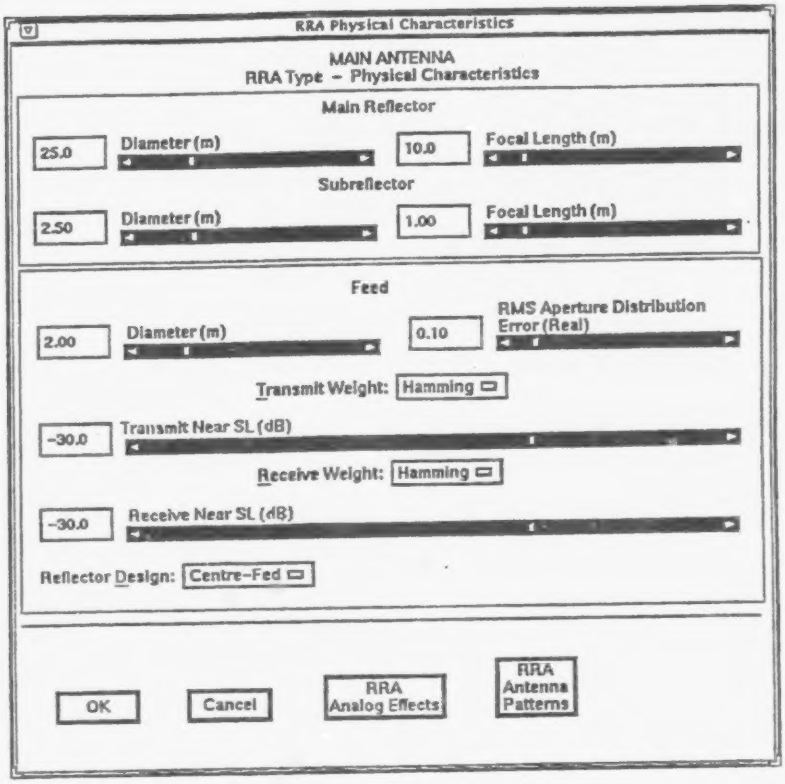
Clicking the "RRA Analog Effects" push button pops up the Analog Effects dialogue (See Section G2.8) for editing the analog effects definition.

Clicking the "RRA Antenna Patterns" push button pops up the RRA Antenna Patterns dialogue (See Section G2.12) for editing the antenna patterns definition.

Fixed ranges will be implemented for the following parameters:
0 <= main reflector diameter <= 100 m
0 <= subreflector focal length <= 100 m
0 <= subreflector diameter <= 100 m
0 <= subreflector focal length <= 100 m
0 <= feed diameter <= 100 m
0 <= feed axial position <= 100 m
0 <= feed grid spacing (dX an dY) <= 0.3 m

Fixed defaults will be implemented for the following parameters:

main reflector diameter	25 m
subreflector focal length	12.5 m
subreflector diameter	2.5 m
subreflector focal length	1.25 m
feed diameter	2.5 m
feed axial position	0 m
feed grid spacing (dX/dY)	0.08 m
reflector elevation	61.8 deg.



RRA Physical Characteristics

MAIN ANTENNA
RRA Type – Physical Characteristics

Main Reflector

Diameter (m): 25.0
Focal Length (m): 10.0

Subreflector

Diameter (m): 2.50
Focal Length (m): 1.00

Feed

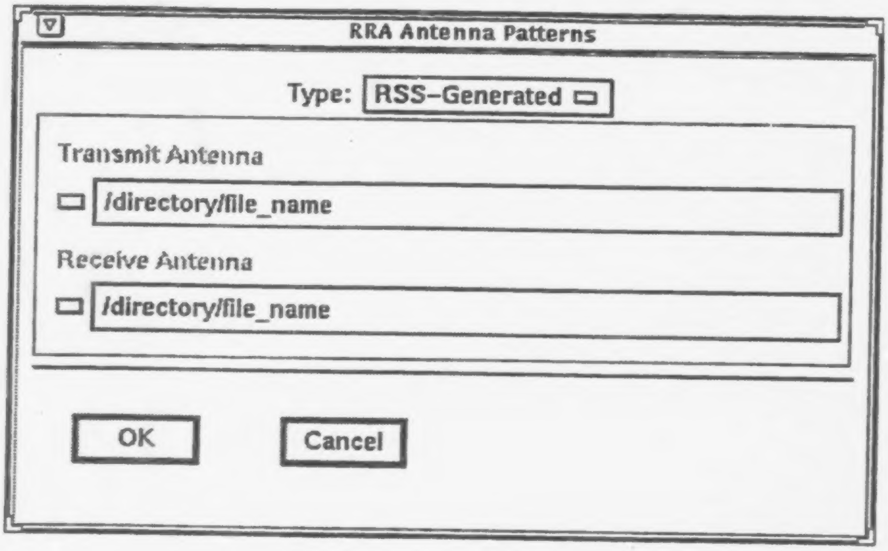
Diameter (m): 2.00
RMS Aperture Distribution Error (Real): 0.10
Transmit Weight: Hamming
Transmit Near SL (dB): -30.0
Receive Weight: Hamming
Receive Near SL (dB): -30.0
Reflector Design: Centre-Fed

crdhRraPhysicalD

G2.12 RRA Antenna Patterns (crdhRraAntennaPatternsD)

– This dialogue box permits the user to make changes to parameters associated with the "RRA Antenna Patterns" definition described in Satellite/Radar HMI parameter Table 2.3.17, 2.3.17.1.

If "RSS-Generated" is selected, the file selection input area is dimmed to disable parameter input. If "User-Generated" is selected, this area is sensitive. When the user clicks the small buttons positioned on the right side of the file name text field, a file selection dialogue is displayed for entering a file name pattern. The text field then displays the file name selected via the file selection dialogue.



RRA Antenna Patterns

Type: RSS-Generated

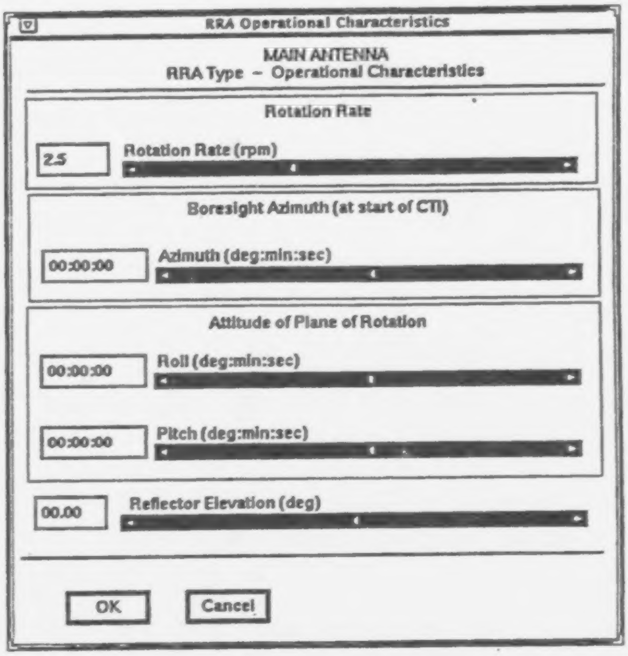
Transmit Antenna: /directory/file_name

Receive Antenna: /directory/file_name

crdhRraAntennaPatternsD

G2.13 RRA Antenna Operational (crdhRraOperationalD)

- This dialogue box permits the user to make changes to parameters associated with the "RRA Antenna Operational" characteristics definition described in Satellite/Radar HMI parameter Table 2.3.13 to 2.3.17.1.



MAIN ANTENNA
RRA Type – Operational Characteristics

Rotation Rate
2.5 Rotation Rate (rpm)

Boresight Azimuth (at start of CTI)
00:00:00 Azimuth (deg:min:sec)

Attitude of Plane of Rotation
00:00:00 Roll (deg:min:sec)
00:00:00 Pitch (deg:min:sec)

00.00 Reflector Elevation (deg)

crdhRraOperationalD

G2.14 Auxiliary Antenna Physical (crdhAuxPhysicalD)

This dialogue box permits the user to make changes to parameters associated with the "Auxiliary Antenna Physical" characteristics definition described in Satellite/Radar HMI parameter Table 2.4 excluding 2.4.3.1, 2.4.4, 2.4.6 and 2.4.7.

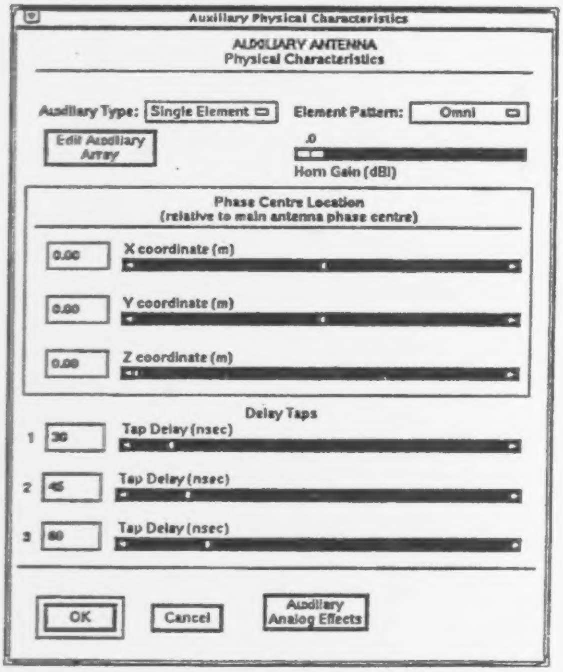
Only the available delay taps are sensitive for entering time delay; the unselected taps are dimmed. The number of delay taps associated with the current auxiliary antenna was defined previously in the "Antenna Selections" dialogue (See Section G.2.4).

If the auxiliary type is "Single Element", then the "Edit Auxiliary Array" button is insensitive. The button is sensitive only if the auxiliary type is "Regular Array" or "Arbitrary Array". Clicking the "Edit Auxiliary Array" push button invokes the array elements definition. The application pops up either the Auxiliary Antenna Regular Array dialogue (See Section G2.15) or the Auxiliary Antenna Arbitrary Array dialogue (See Section G2.16) corresponding to the specified array type.

Clicking the "Auxiliary Analog Effects" push button pops up the Analog Effects dialogue (See Section G2.8) for editing the analog effects definition.

Fixed defaults will be implemented for the following parameters:

- time delay:	30 ns for tap 1; 45 ns for tap 2; 60 ns for tap 3.



Auxiliary Physical Characteristics

AUXILIARY ANTENNA
Physical Characteristics

Auxiliary Type: Single Element
Element Pattern: Omni

Horn Gain (dBI): .0

Phase Centre Location
(relative to main antenna phase centre)

X coordinate (m): 0.00
Y coordinate (m): 0.00
Z coordinate (m): 0.00

Delay Taps

1 Tap Delay (nsec): 30
2 Tap Delay (nsec): 45
3 Tap Delay (nsec): 60

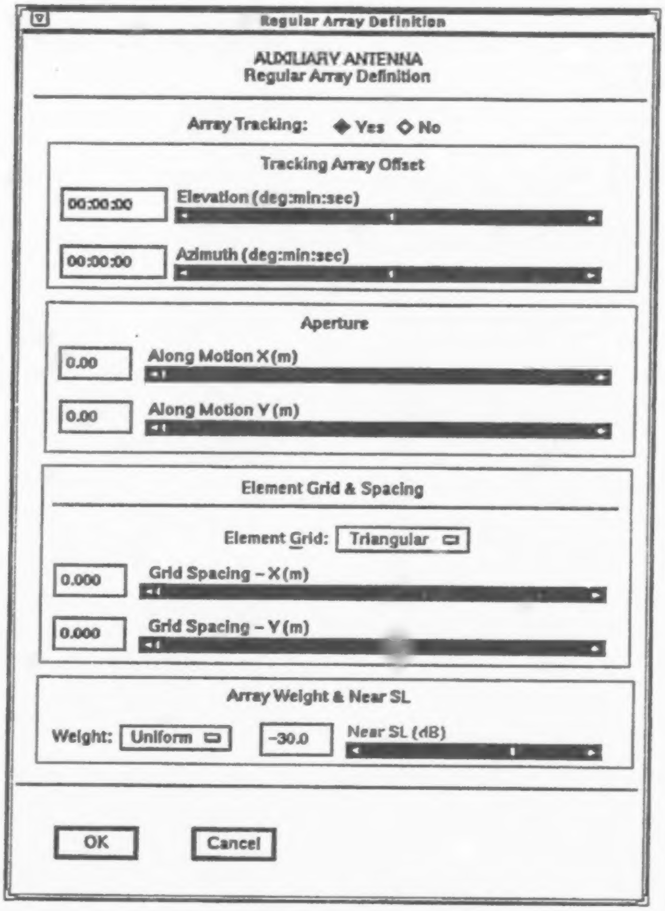
crdhAuxPhysicalD

G2.15 Auxiliary Antenna Regular Array (crdhAuxRegularArrayD)

This dialogue box permits the user to make changes to parameters associated with the "Regular Array" definition described in Satellite/Radar HMI parameter Table 2.4.6.

Fixed ranges will be implemented for the following parameters:
0 <= grid spacing (dX and dY) <= 0.3 m

Fixed defaults will be implemented for the following parameters:
grid spacing (dX and dY) 0.08 m



Regular Array Definition

AUXILIARY ANTENNA
Regular Array Definition

Array Tracking: ● Yes ○ No

Tracking Array Offset
00:00:00 Elevation (deg:min:sec)
00:00:00 Azimuth (deg:min:sec)

Aperture
0.00 Along Motion X (m)
0.00 Along Motion Y (m)

Element Grid & Spacing
Element Grid: Triangular
0.000 Grid Spacing – X (m)
0.000 Grid Spacing – Y (m)

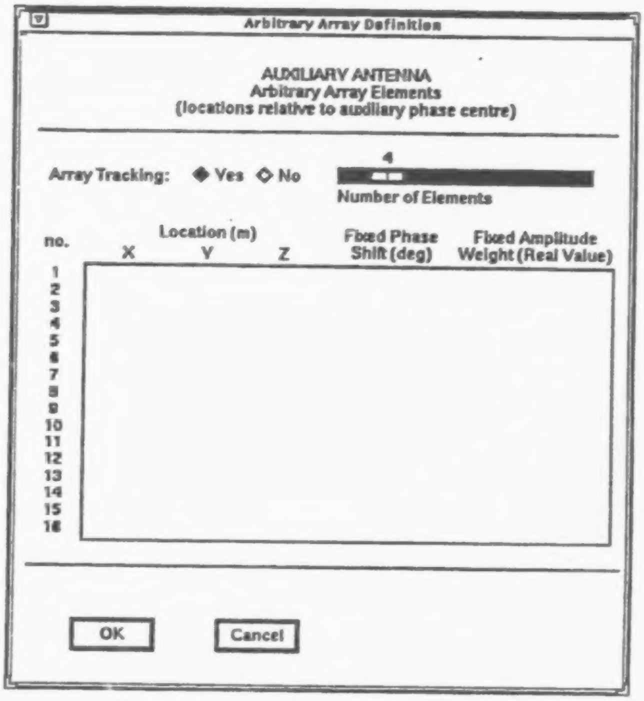
Array Weight & Near SL
Weight: Uniform
-30.0 Near SL (dB)

crdhAuxRegularArrayD

G2.16 Auxiliary Antenna Arbitrary Array (crdhAuxArbitraryArrayD)

This dialogue box permits the user to make changes to parameters associated with the "Arbitrary Array" definition described in Satellite/Radar HMI parameter Table 2.4.7.

Smart defaults will be implemented for the following parameters:
Element location (X and Y)- circular placement



Arbitrary Array Definition

AUXILIARY ANTENNA
Arbitrary Array Elements
(locations relative to auxiliary phase centre)

Array Tracking: ◆ Yes ◇ No

Number of Elements: 4

no.	Location (m) X	Y	Z	Fixed Phase Shift (deg)	Fixed Amplitude Weight (Real Value)
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					

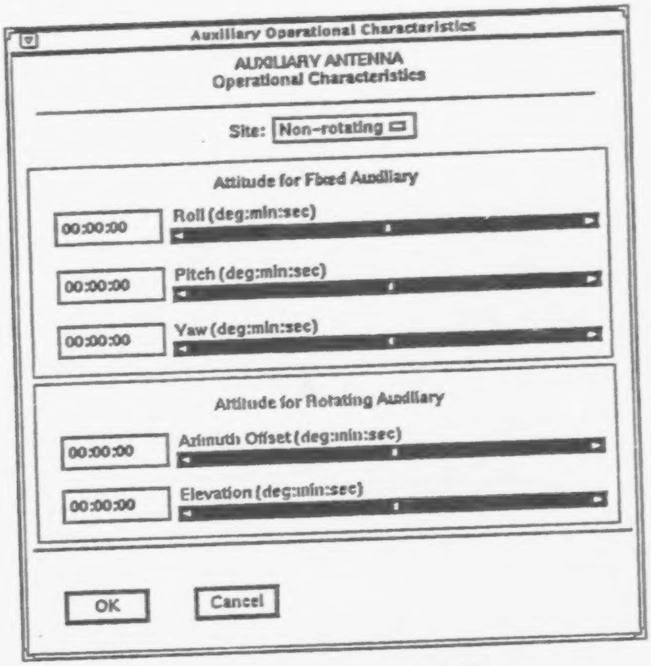
crdhAuxArbitraryArrayD

G2.17 Auxiliary Antenna Operational (crdhAuxOperationalD)

This dialogue box permits the user to make changes to parameters associated with the "Auxiliary Antenna Operational" definition described in Satellite/Radar HMI parameter Table 2.4.3.1 and 2.4.4.

Fixed defaults will be implemented for the following parameters:

attitude (roll, pitch, yaw)	0 deg.
elevation	61.8 deg.


Auxiliary Operational Characteristics
AUXILIARY ANTENNA
Operational Characteristics
Attitude for Fixed Auxiliary
Attitude for Rotating Auxiliary
OK
Cancel

Site: Non-rotating

Roll (deg:min:sec): 00:00:00

Pitch (deg:min:sec): 00:00:00

Yaw (deg:min:sec): 00:00:00

Azimuth Offset (deg:min:sec): 00:00:00

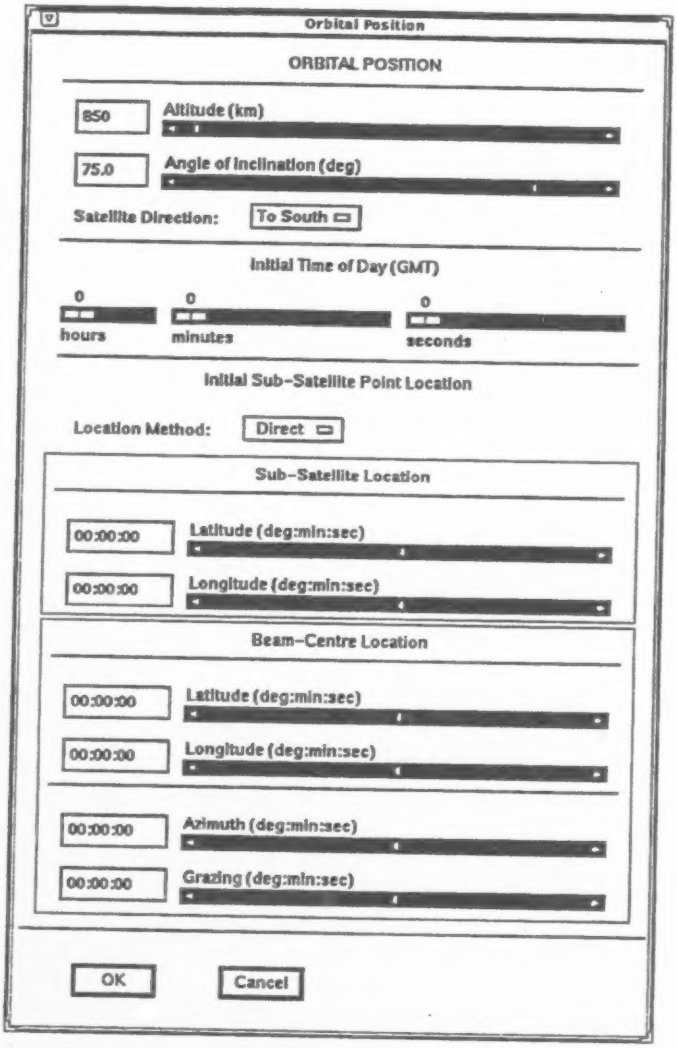
Elevation (deg:min:sec): 00:00:00

crdhAuxOperationalD

G2.18 Orbital Position (crdhOrbitalPostionD)

This dialogue box permits the user to make changes to parameters associated with the "Orbital Position" definition described in Satellite/Radar HMI parameter Table 2.1.

According to the location method selected ("direct" or "derived"), either the "sub-satellite location" input area or the "beam-centre location" input area becomes sensitive for input.



ORBITAL POSITION

Altitude (km): 850

Angle of Inclination (deg): 75.0

Satellite Direction: To South

Initial Time of Day (GMT)

hours: 0
minutes: 0
seconds: 0

Initial Sub-Satellite Point Location

Location Method: Direct

Sub-Satellite Location

Latitude (deg:min:sec): 00:00:00
Longitude (deg:min:sec): 00:00:00

Beam-Centre Location

Latitude (deg:min:sec): 00:00:00
Longitude (deg:min:sec): 00:00:00

Azimuth (deg:min:sec): 00:00:00
Grazing (deg:min:sec): 00:00:00

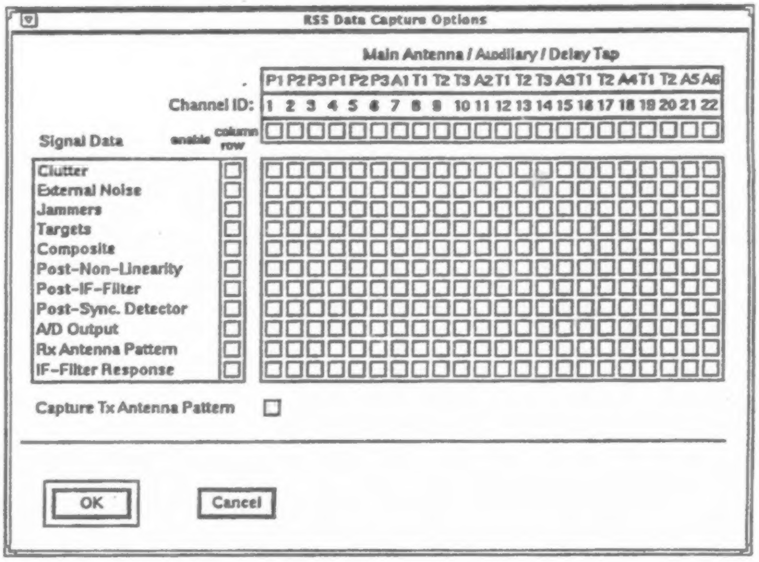
crdhOrbitalPositionD

G2.19 Data Capture Options (crdhCaptureOptionsD)

This dialogue box permits the user to make changes to parameters associated with the "RSS Data Capture Options" definition described in Satellite/Radar HMI parameter Table 6.1 to 6.11.

Main antenna, phase centres, auxiliaries and delay taps with their channel IDs are displayed on the annotation area. Toggling of the push button in the grid button area specifies the data capture option: "on" (pressed or filled) indicating enabled or "off" (unpressed) indicating disabled. Note that "RX Antenna Pattern" is not applicable to delay taps and therefore, the associated push buttons are inactive.

The push buttons beside the signal data text are used for capturing all channels for a specific signal type; i.e. toggling one of these buttons enables/disables an entire row of buttons. The buttons under the channel annotation have similar function except that they enable/disable an entire column of buttons: "on" to capture all signal data for one channel or "off" not to capture.



		P1	P2	P3	P1	P2	P3	A1	T1	T2	T3	A2	T1	T2	T3	A3	T1	T2	A4	T1	T2	A5	A6
Channel ID:		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Signal Data	enable column row	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Clutter	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
External Noise	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Jammers	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Targets	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Composite	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Post-Non-Linearity	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Post-IF-Filter	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Post-Sync. Detector	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
A/D Output	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Rx Antenna Pattern	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
IF-Filter Response	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Capture Tx Antenna Pattern ☐

crdhCaptureOptionsD

G2.20 Data Reduction Options (crdhDataReductionD)

This dialogue box permits the user to make changes to parameters associated with the "Data Reduction" definition described in Satellite/Radar HMI parameter Table 6.12.1, 6.12.2, 6.12.3.

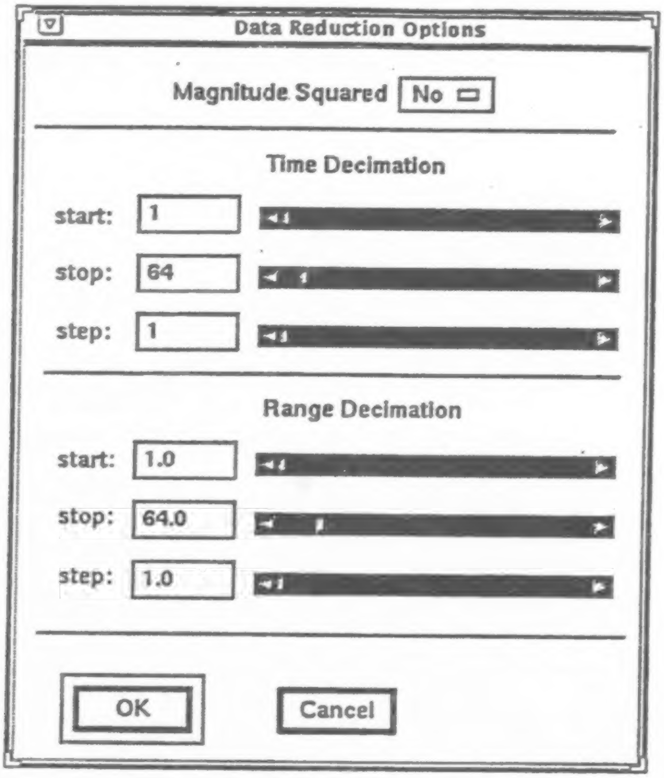
These options apply to all captured signals.

Fixed ranges will be implemented for the following parameters:
1<= time decimation <= 1500,1500,1499
1 <= range decimation <= 512,512,511

Fixed defaults will be implemented for the following parameters:

time decimation	1,64,1
range decimation	1,64,1

Floating point inputs are used for range decimation since some RSS products are oversampled in range


Data Reduction Options

Magnitude Squared: No

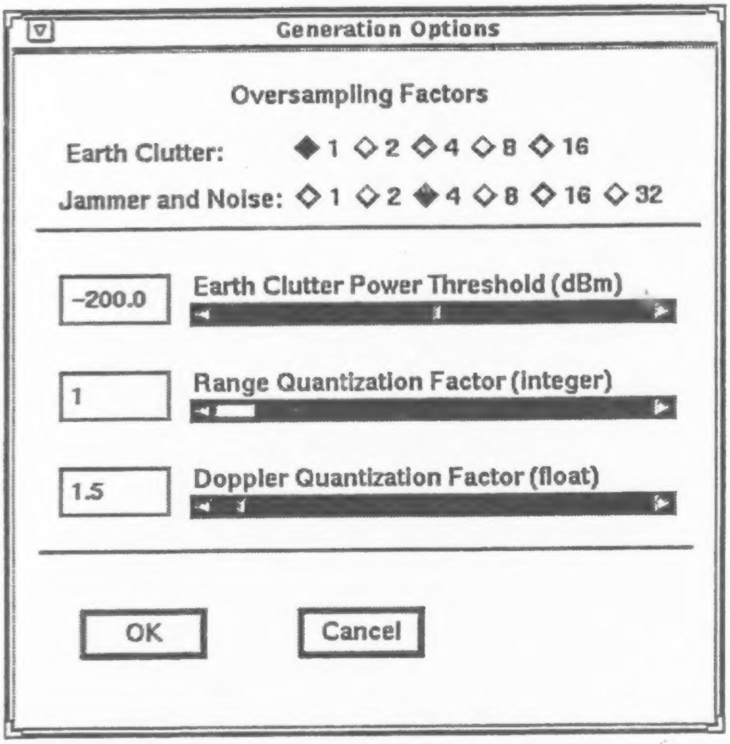
Time Decimation
start: 1
stop: 64
step: 1

Range Decimation
start: 1.0
stop: 64.0
step: 1.0

crdhDataReductionD

G2.21 Generation Options (crdhGenerationOptionsD)

This dialogue box permits the user to make changes to parameters associated with the "Generation" definition described in Satellite/Radar HMI parameter Table 7. Smart range will be implemented for the Earth Clutter Oversampling Factor to ensure that it is not greater than the Jammer and Noise Oversampling Factor.



Generation Options

Oversampling Factors

Earth Clutter: ◆ 1 ◇ 2 ◇ 4 ◇ 8 ◇ 16

Jammer and Noise: ◇ 1 ◇ 2 ◆ 4 ◇ 8 ◇ 16 ◇ 32

-200.0 Earth Clutter Power Threshold (dBm)

1 Range Quantization Factor (integer)

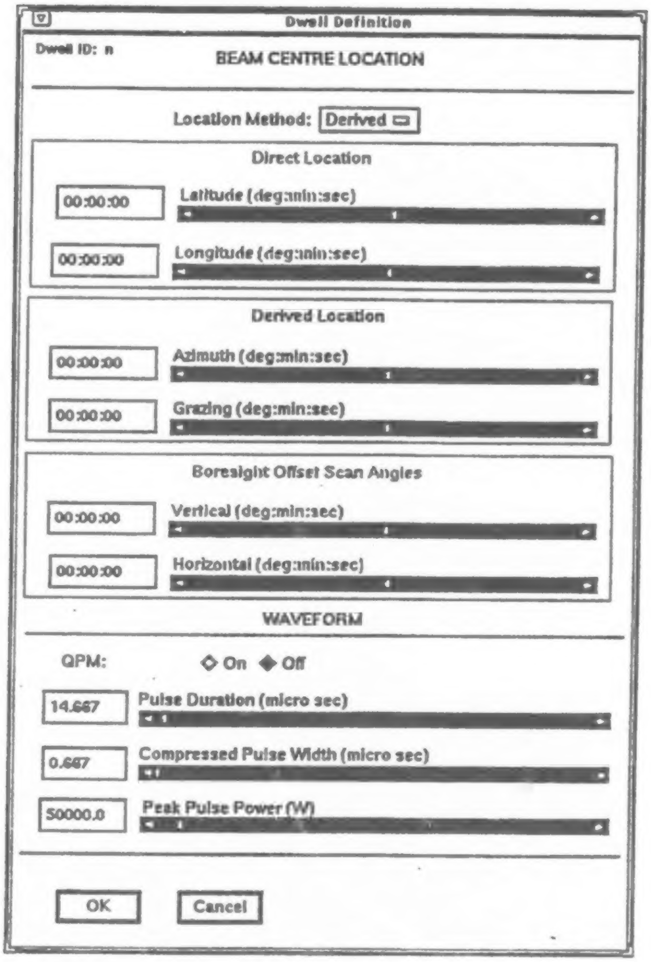
1.5 Doppler Quantization Factor (float)

crdhGenerationOptionsD

G2.22 Dwell Definition (crdhDwellDefD)

This dialogue box permits the user to make changes to parameters associated with the "Dwell" definition described in Satellite/Radar HMI parameter Table 3, excluding 3.3. This dialogue contains two areas: beam pointing and waveform areas.

Fixed ranges will be implemented for the following parameters:
0 <= pulse duration <= 1000 micro sec
0 < = compressed pulse width <= 1000 micro sec



Dwell Definition

Dwell ID: n

BEAM CENTRE LOCATION

Location Method: Derived

Direct Location

Latitude (deg:min:sec): 00:00:00

Longitude (deg:min:sec): 00:00:00

Derived Location

Azimuth (deg:min:sec): 00:00:00

Grazing (deg:min:sec): 00:00:00

Boresight Offset Scan Angles

Vertical (deg:min:sec): 00:00:00

Horizontal (deg:min:sec): 00:00:00

WAVEFORM

QPM: ◇ On ◆ Off

Pulse Duration (micro sec): 14.667

Compressed Pulse Width (micro sec): 0.667

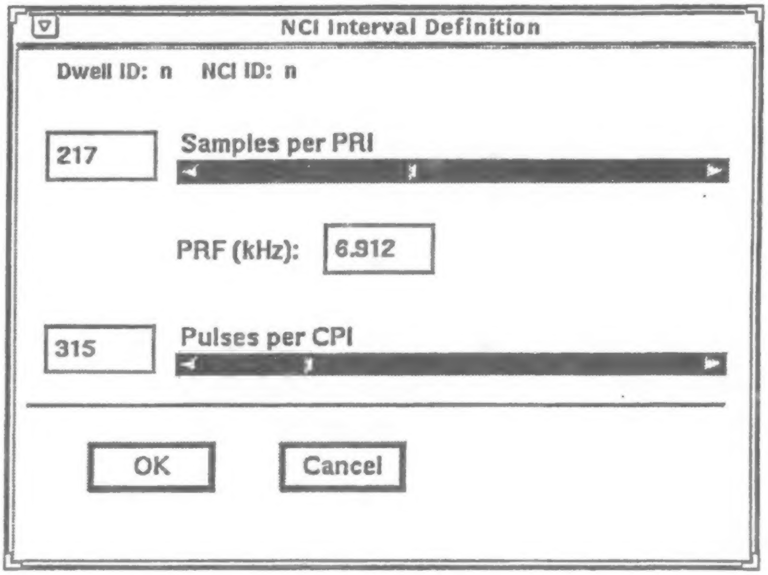
Peak Pulse Power (W): 50000.0

crdhDwellDefD

G2.23 NCI Definition (crdhNciDefD)

This dialogue box permits the user to make changes to parameters associated with the "NCI Interval" definition described in Satellite/Radar HMI parameter Table 4, excluding 4.2.

Fixed ranges will be implemented for the following parameters:
10 < = samples per PRI <= 512



Dwell ID: n NCI ID: n

Samples per PRI: 217

PRF (kHz): 6.912

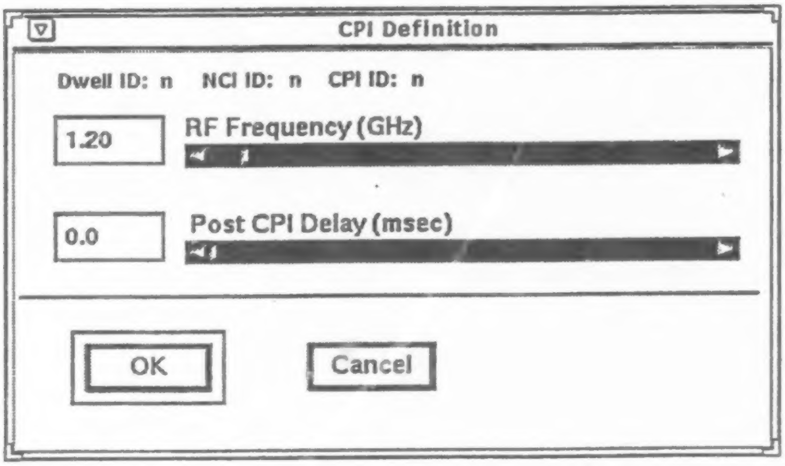
Pulses per CPI: 315

crdhNciDefD

G2.24 CPI Definition (crdhCpiDefD)

- This dialogue box permits the user to make changes to parameters associated with the "CPI Interval" definition described in Satellite/Radar HMI parameter Table 5.

Fixed ranges will be implemented for the following parameters:
1 <= RF frequency<= 4 GHz



CPI Definition

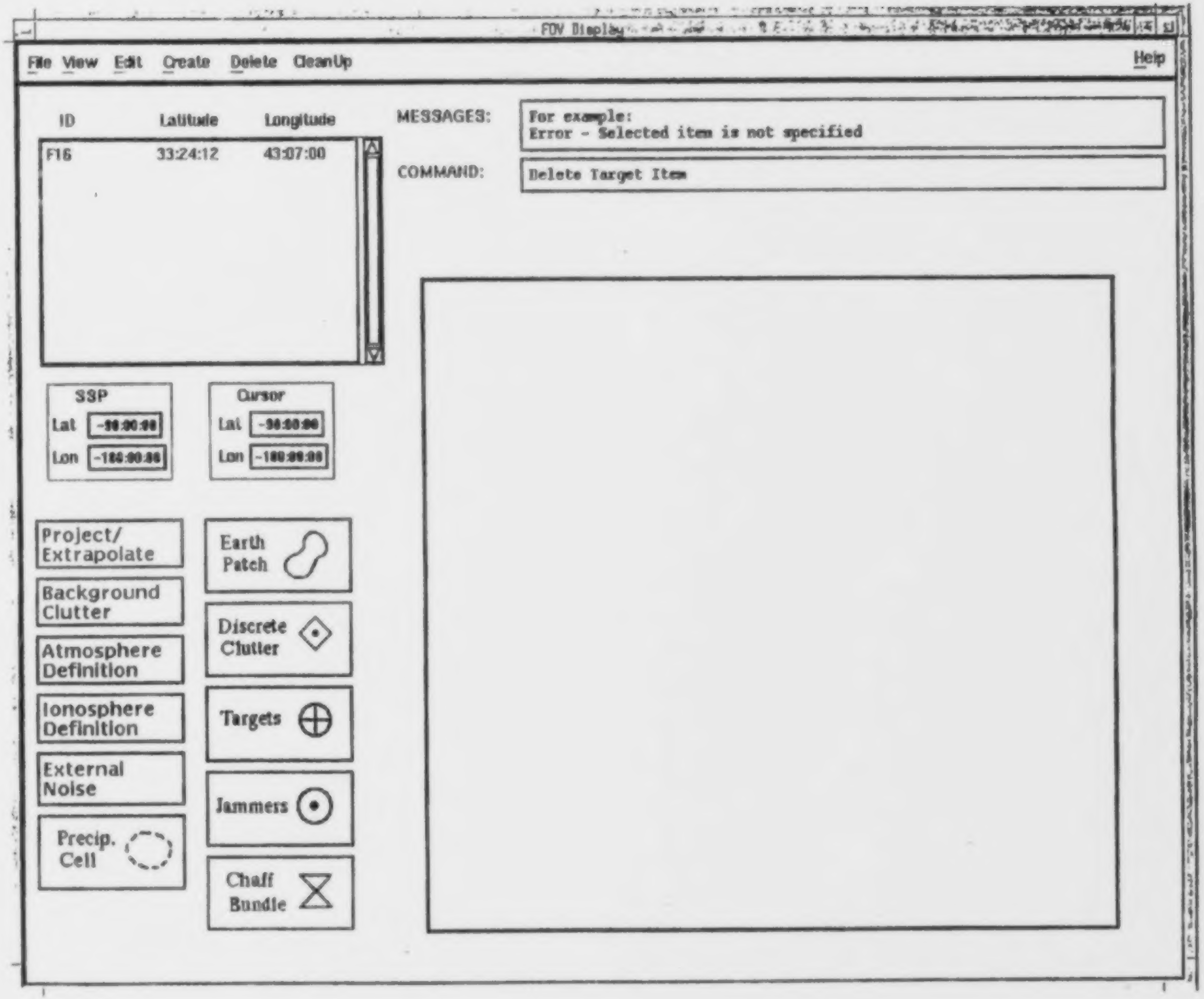
Dwell ID: n NCI ID: n CPI ID: n

RF Frequency (GHz): 1.20

Post CPI Delay (msec): 0.0

crdhCpiDefD

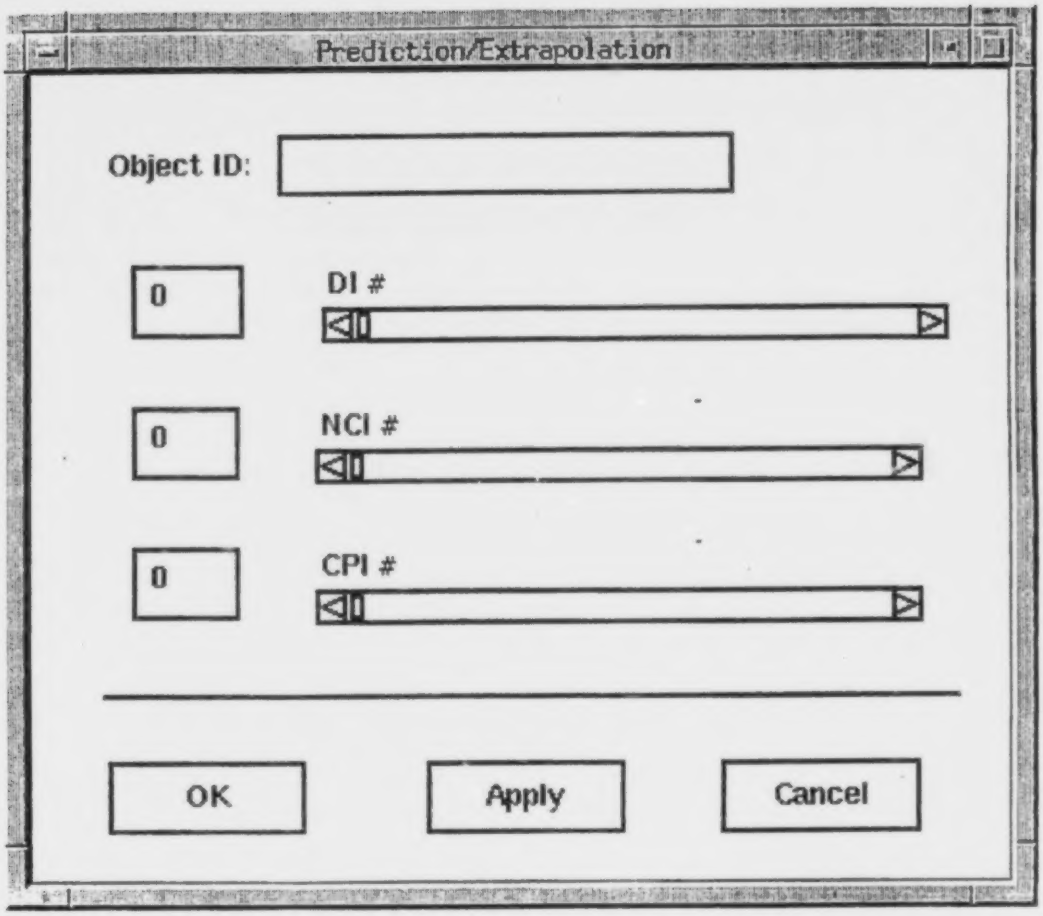
G3 ENV DEFINITION (CEDH) CSU


FDV Display
File
View
Edit
Create
Delete
CleanUp
Help
SSP
Lat
Lon
Cursor
Lat
Lon
Project/ Extrapolate
Background Clutter
Atmosphere Definition
Ionosphere Definition
External Noise
Precip. Cell
Earth Patch
Discrete Clutter
Targets
Jammers
Chaff Bundle

ID	Latitude	Longitude
F16	33:24:12	43:07:00

MESSAGES: For example:
Error - Selected item is not specified

COMMAND: Delete Target Item



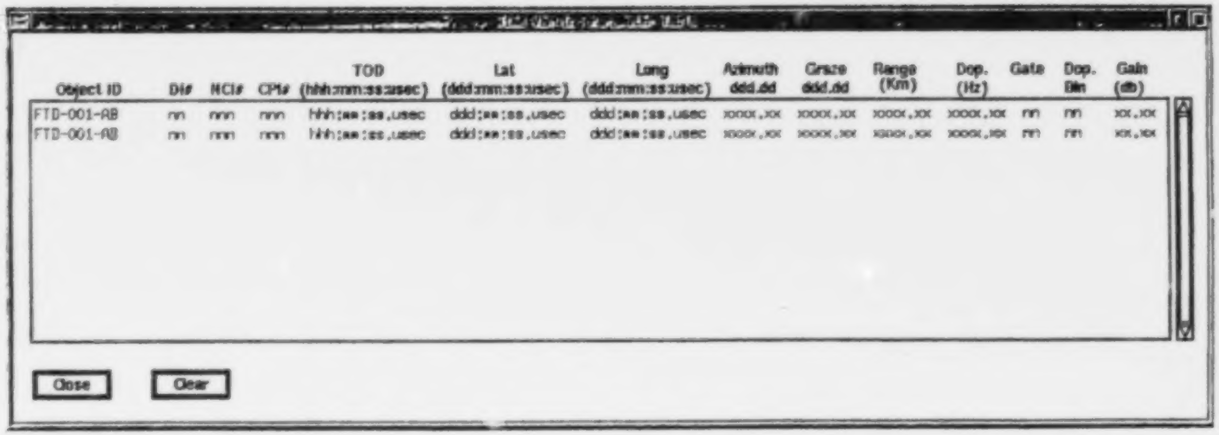
Prediction/Extrapolation

Object ID:

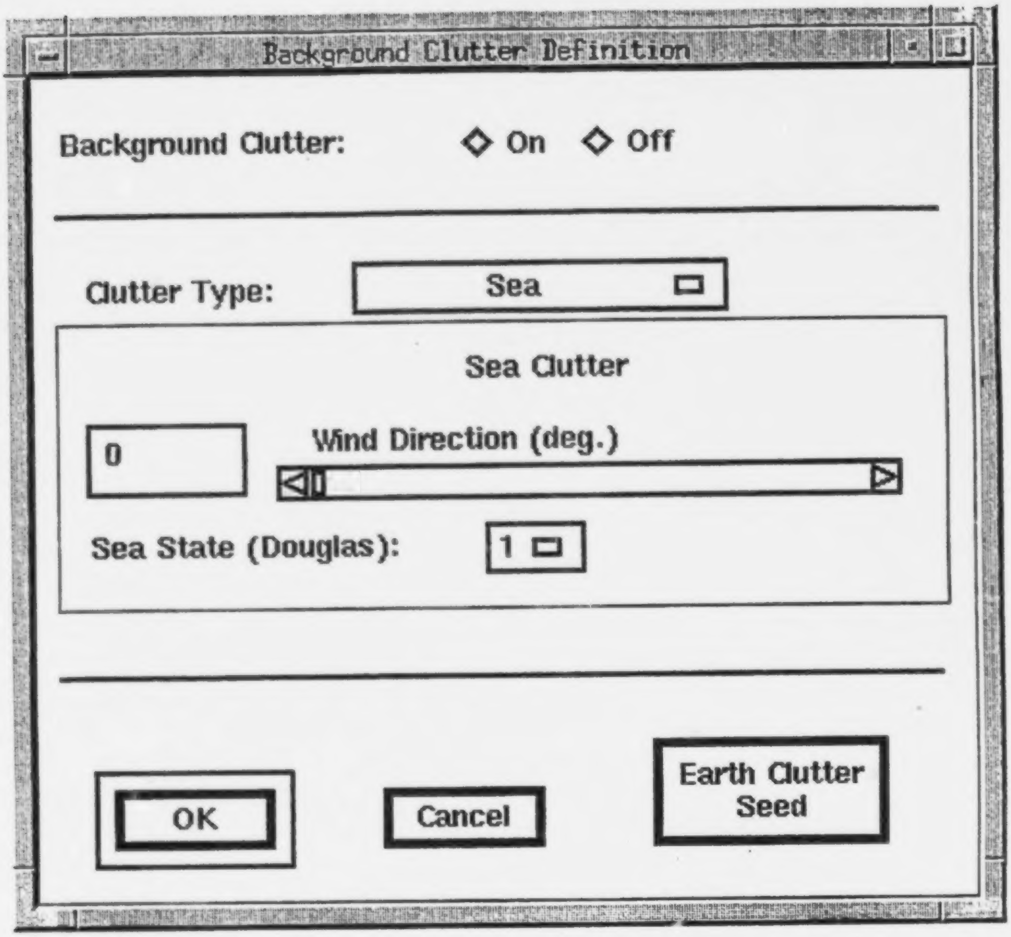
DI # 0

NCI # 0

CPI # 0



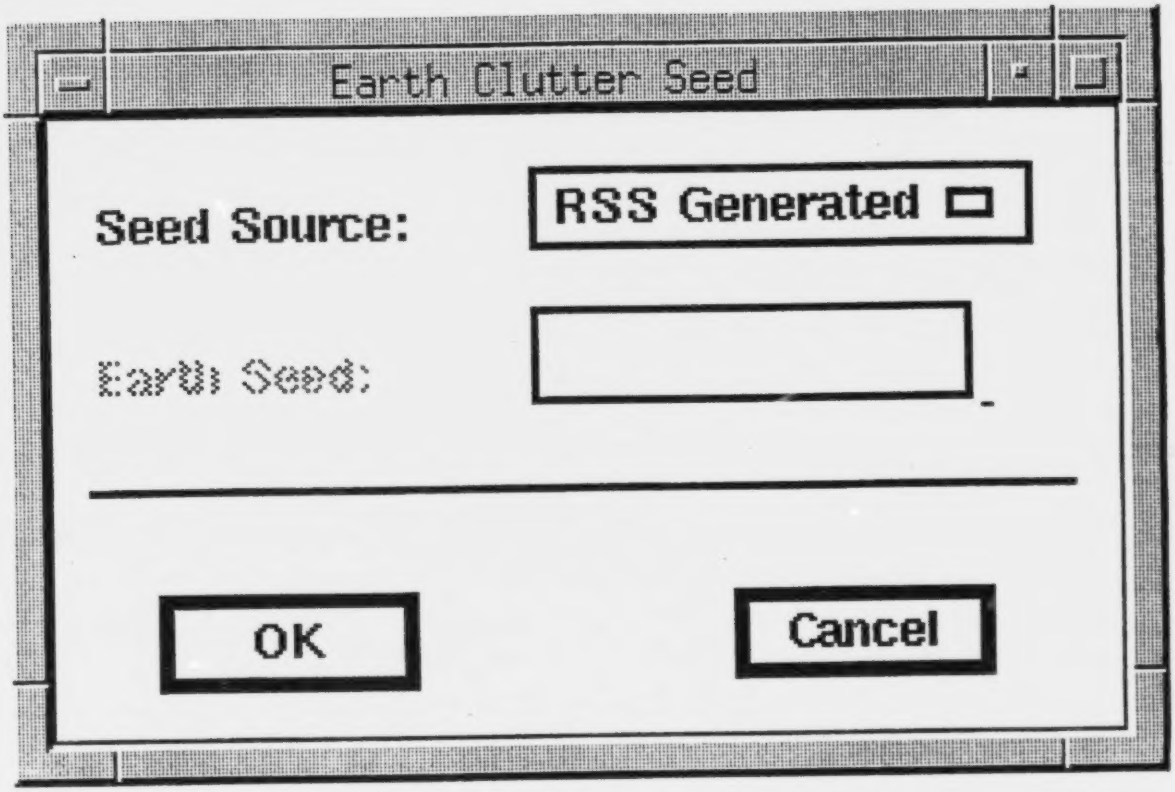
Object ID	DI#	NCI#	CPI#	TOD (hhh:mm:ss:usec)	Lat (ddd:mm:ss:usec)	Long (ddd:mm:ss:usec)	Azimuth ddd.dd	Graze ddd.dd	Range (Km)	Dop. (Hz)	Gate	Dop. Bin	Gain (db)
FTD-001-AB	nn	nnn	nnn	hhh:mm:ss,usec	ddd:mm:ss,usec	ddd:mm:ss,usec	xxxx,xx	xxxx,xx	xxxx,xx	xxxx,xx	nn	nn	xx,xx
FTD-001-AB	nn	nnn	nnn	hhh:mm:ss,usec	ddd:mm:ss,usec	ddd:mm:ss,usec	xxxx,xx	xxxx,xx	xxxx,xx	xxxx,xx	nn	nn	xx,xx


Background Clutter Definition
Background Clutter:
◇ On
◇ Off
Sea Clutter
OK
Cancel
Earth Clutter Seed

Clutter Type: Sea

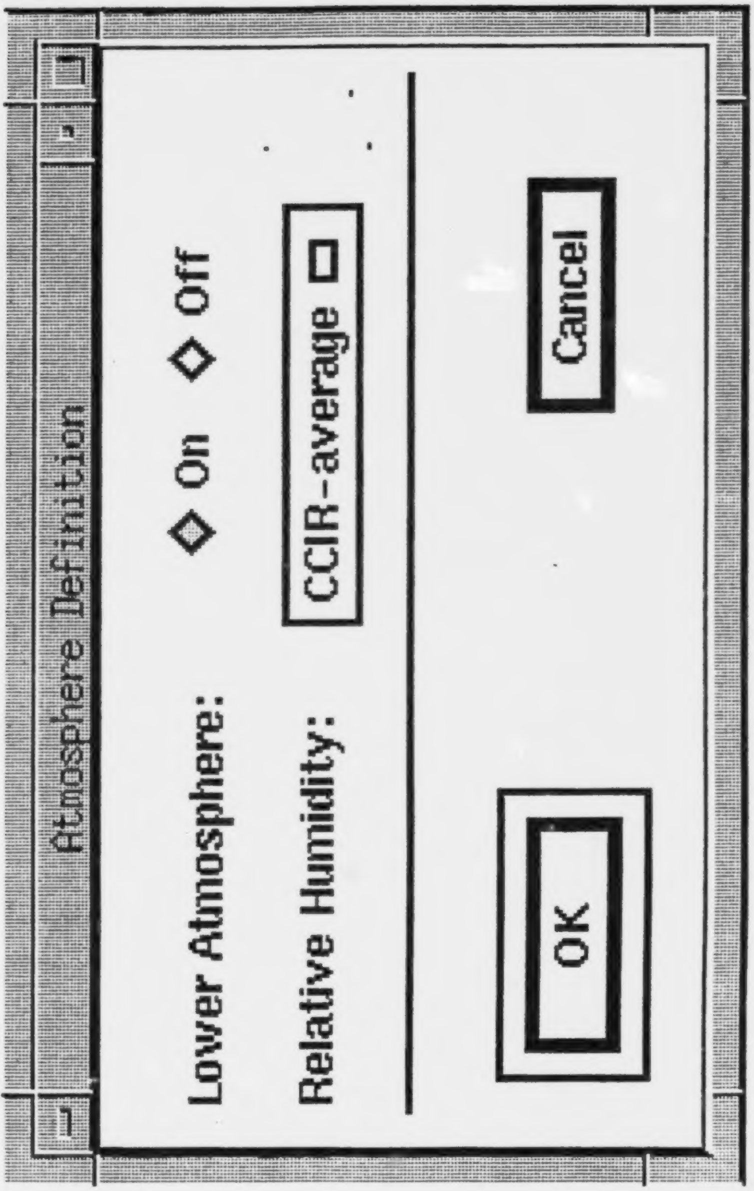
Wind Direction (deg.): 0

Sea State (Douglas): 1


Earth Clutter Seed
OK
Cancel

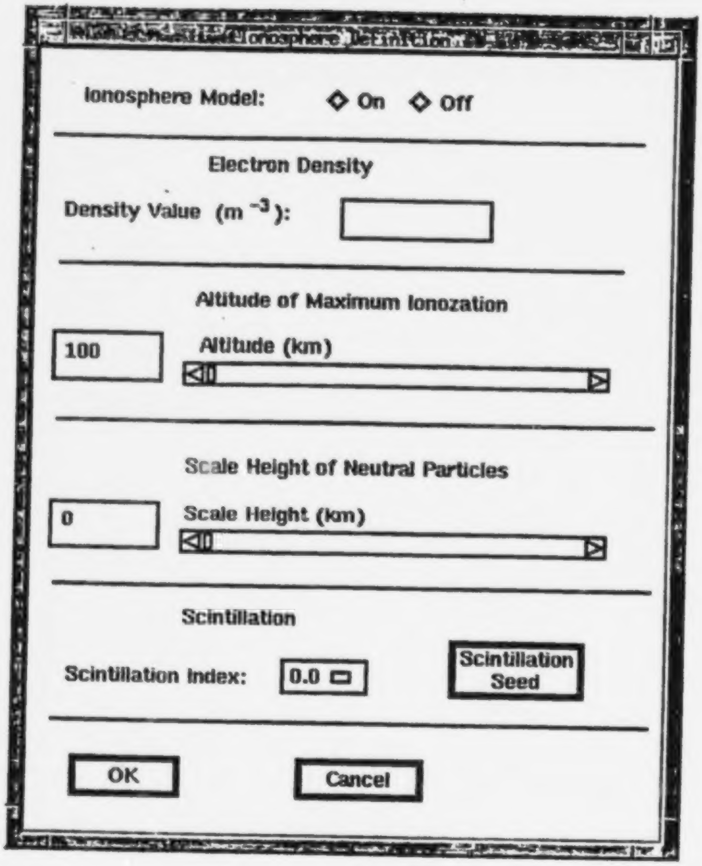
Seed Source: RSS Generated

Earth Seed:


Atmosphere Definition
OK
Cancel

Lower Atmosphere: ◆ On ◇ Off

Relative Humidity: CCIR-average ☐


Ionosphere Definition
Electron Density
Altitude of Maximum Ionozation
Scale Height of Neutral Particles
Scintillation
Scintillation Seed
OK
Cancel

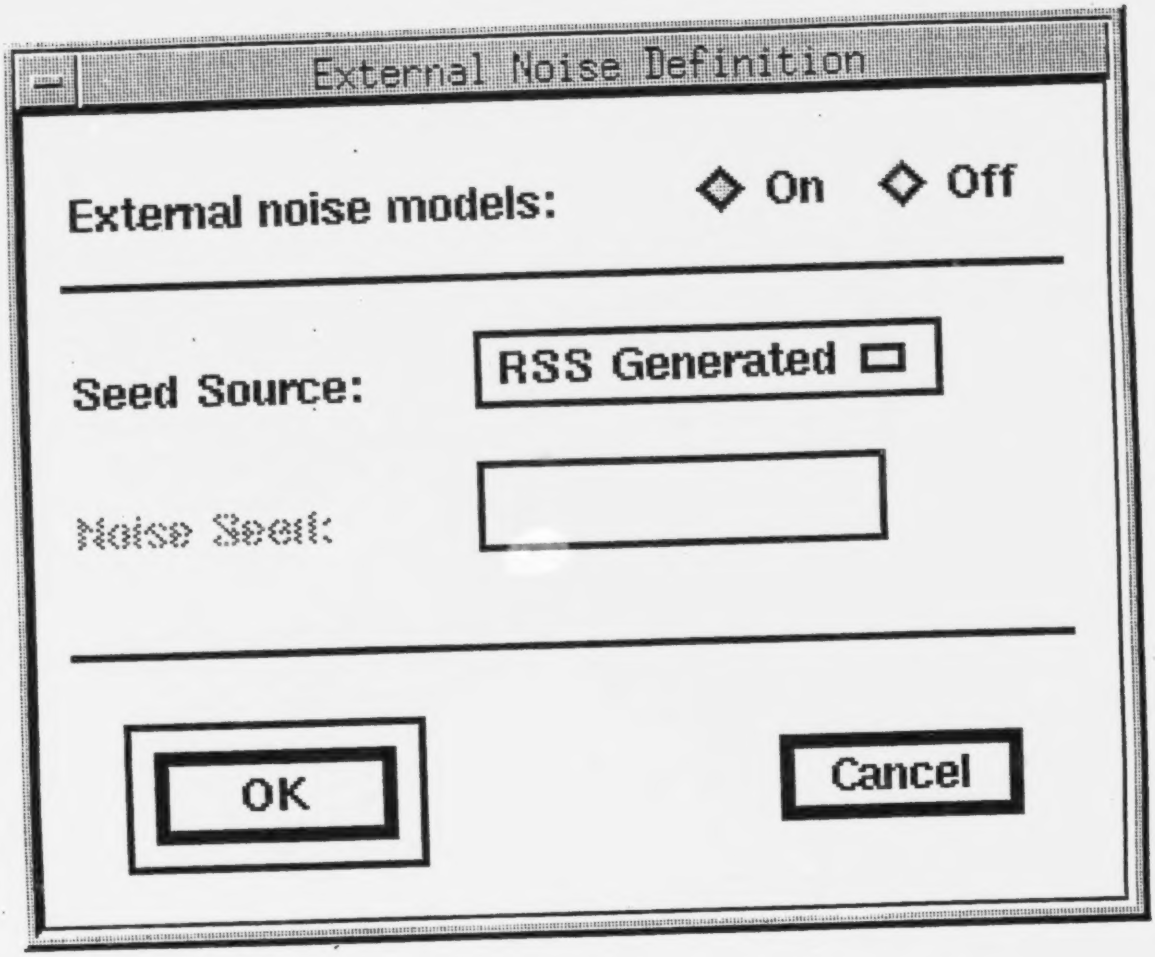
Ionosphere Model: ○ On ○ Off

Density Value (m^{-3}):

Altitude (km): 100

Scale Height (km): 0

Scintillation Index: 0.0

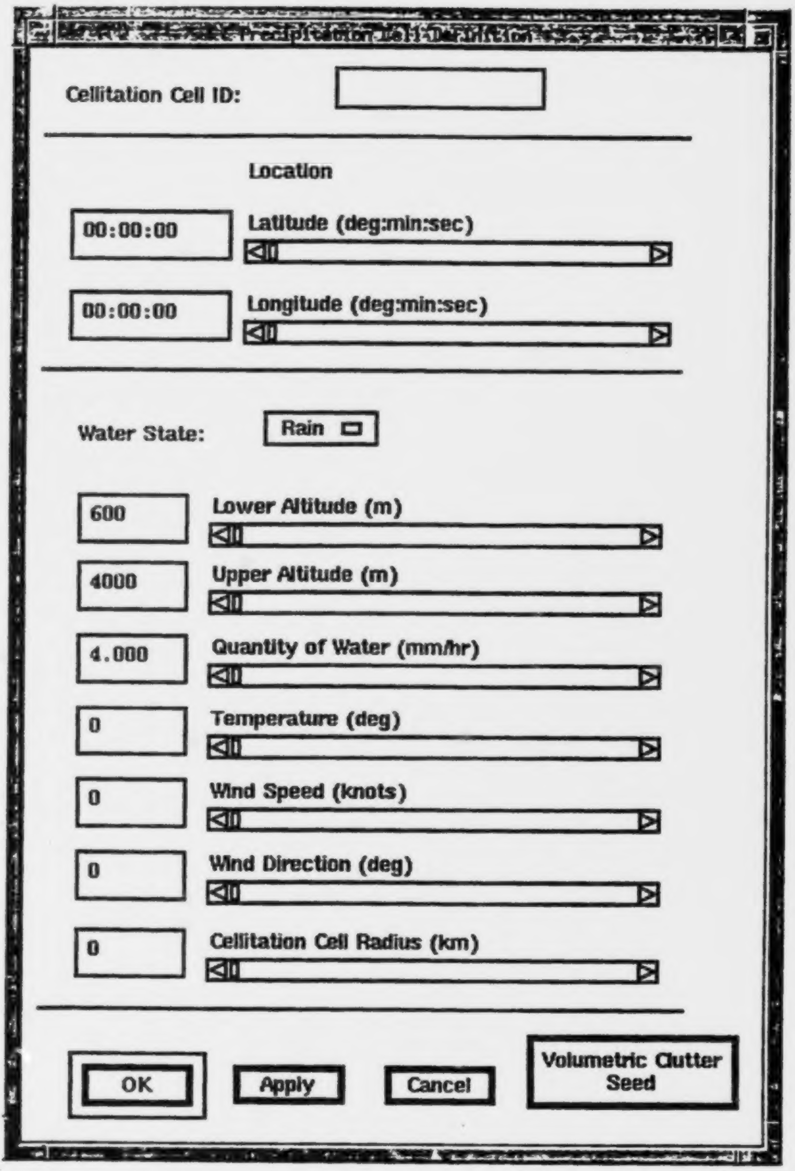


External Noise Definition

External noise models: ◆ On ◇ Off

Seed Source: RSS Generated

Noise Seed:


[illegible] Precipitation Cell Definition [illegible]
Location
OK
Apply
Cancel
Volumetric Clutter Seed

Cellitation Cell ID:

Latitude (deg:min:sec): 00:00:00

Longitude (deg:min:sec): 00:00:00

Water State: Rain ☐

Lower Altitude (m): 600

Upper Altitude (m): 4000

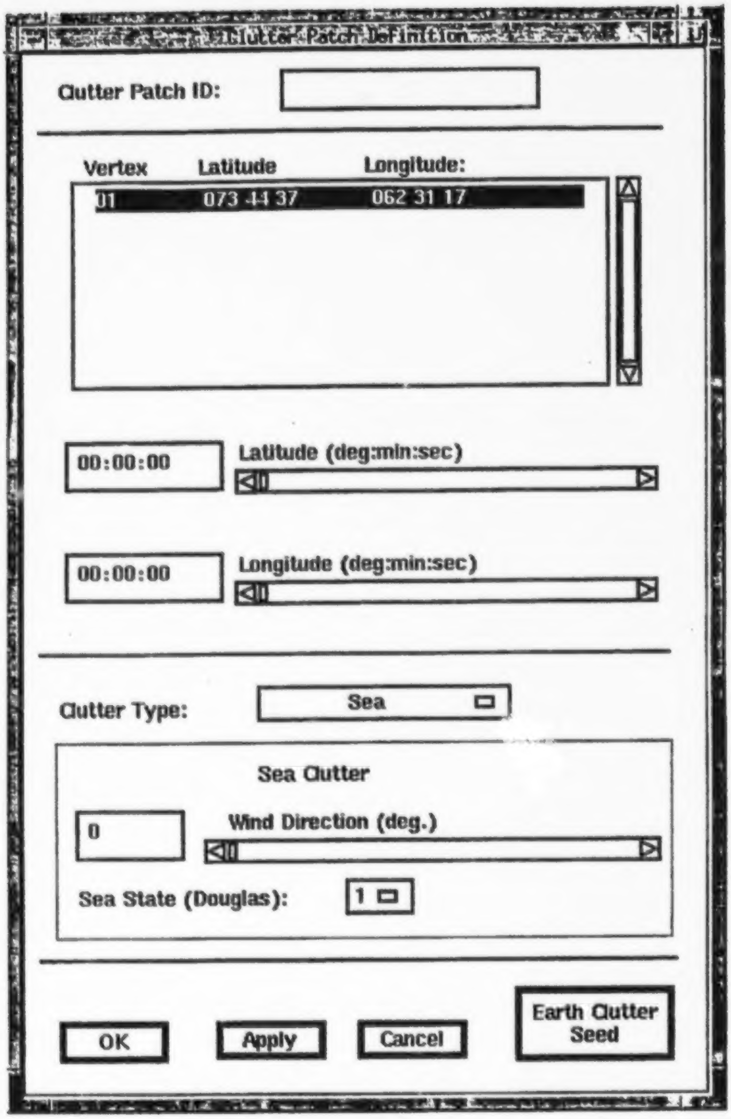
Quantity of Water (mm/hr): 4.000

Temperature (deg): 0

Wind Speed (knots): 0

Wind Direction (deg): 0

Cellitation Cell Radius (km): 0


Clutter Patch Definition
OK
Apply
Cancel
Earth Clutter Seed

Clutter Patch ID:

Vertex	Latitude	Longitude:
01	073 44 37	062 31 17

00:00:00 Latitude (deg:min:sec)

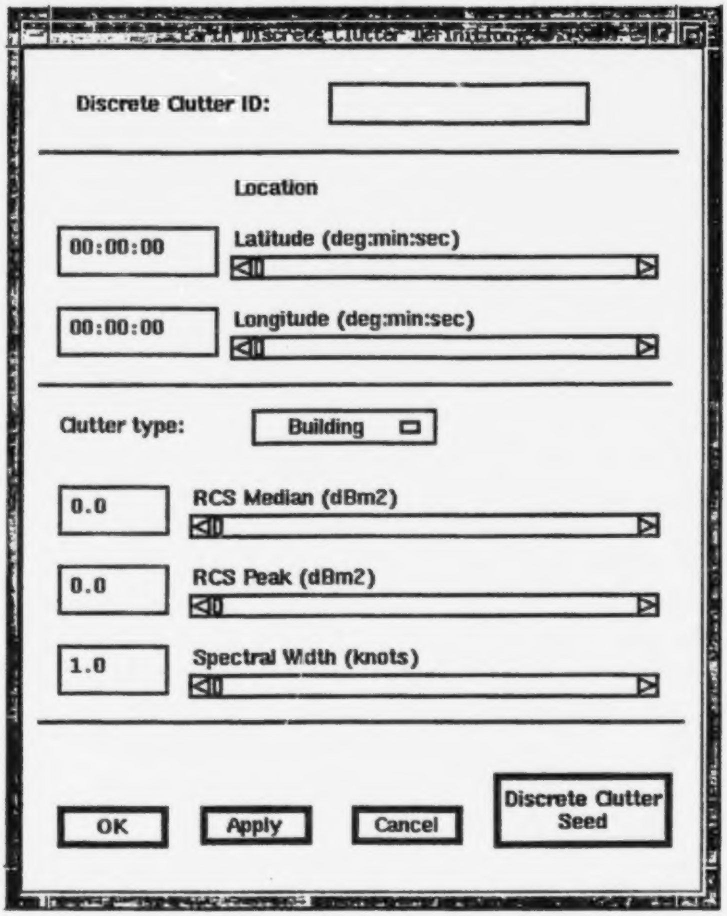
00:00:00 Longitude (deg:min:sec)

Clutter Type: Sea

Sea Clutter

0 Wind Direction (deg.)

Sea State (Douglas): 1


Earth Discrete Clutter Definition [illegible]
OK
Apply
Cancel
Discrete Clutter Seed

Discrete Clutter ID:

Location

Latitude (deg:min:sec): 00:00:00

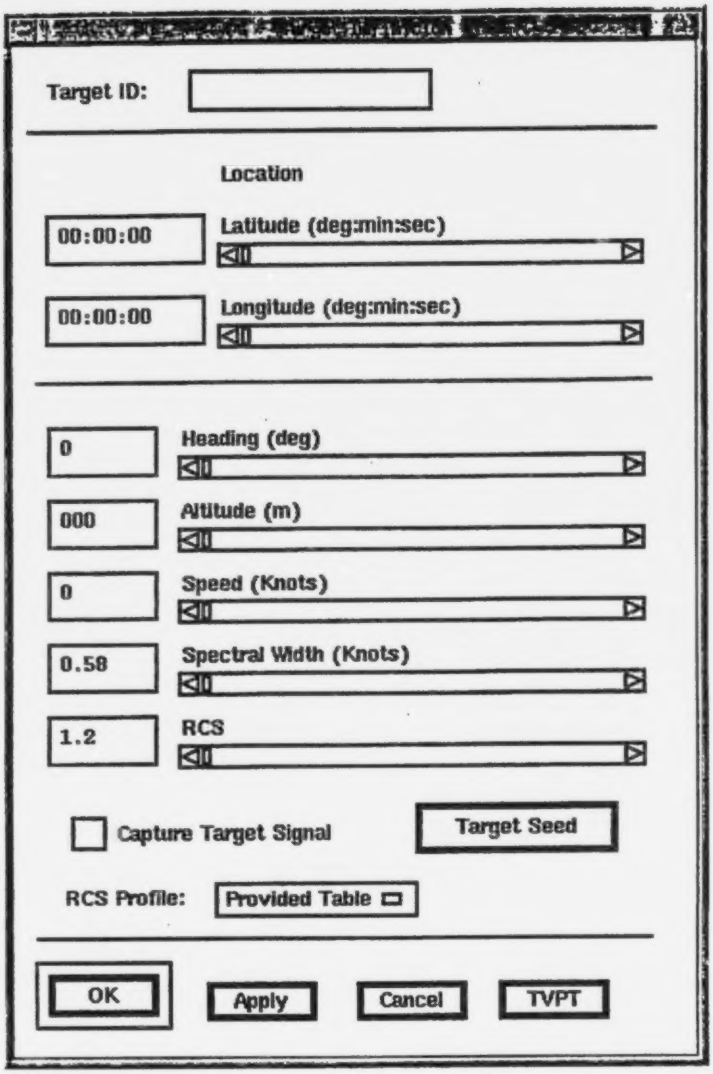
Longitude (deg:min:sec): 00:00:00

Clutter type: Building

RCS Median (dBm2): 0.0

RCS Peak (dBm2): 0.0

Spectral Width (knots): 1.0



Target ID:

Location

Latitude (deg:min:sec): 00:00:00

Longitude (deg:min:sec): 00:00:00

Heading (deg): 0

Altitude (m): 000

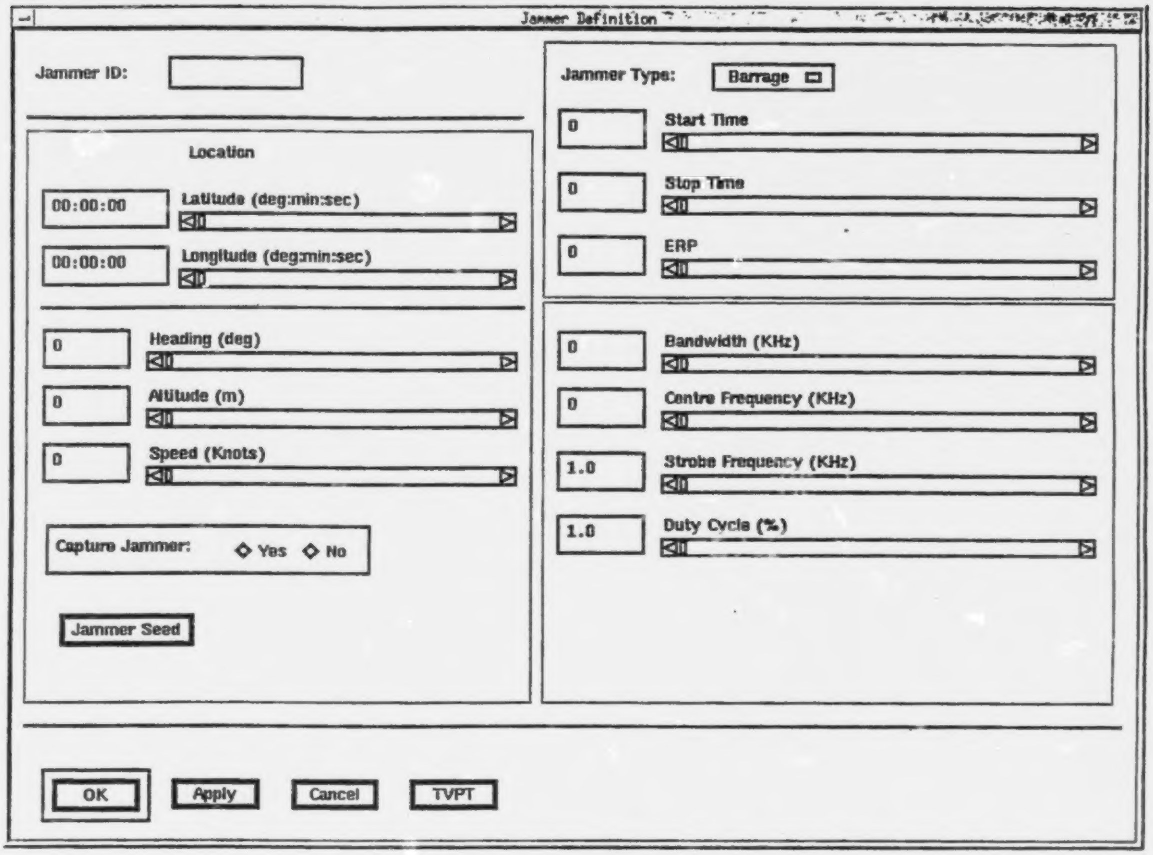
Speed (Knots): 0

Spectral Width (Knots): 0.58

RCS: 1.2

☐ Capture Target Signal

RCS Profile: Provided Table


Jammer Definition
Location
Jammer Seed
OK
Apply
Cancel
TVPT

Jammer ID:

Latitude (deg:min:sec): 00:00:00

Longitude (deg:min:sec): 00:00:00

Heading (deg): 0

Altitude (m): 0

Speed (Knots): 0

Capture Jammer: ◇ Yes ◇ No

Jammer Type: Barrage

Start Time: 0

Stop Time: 0

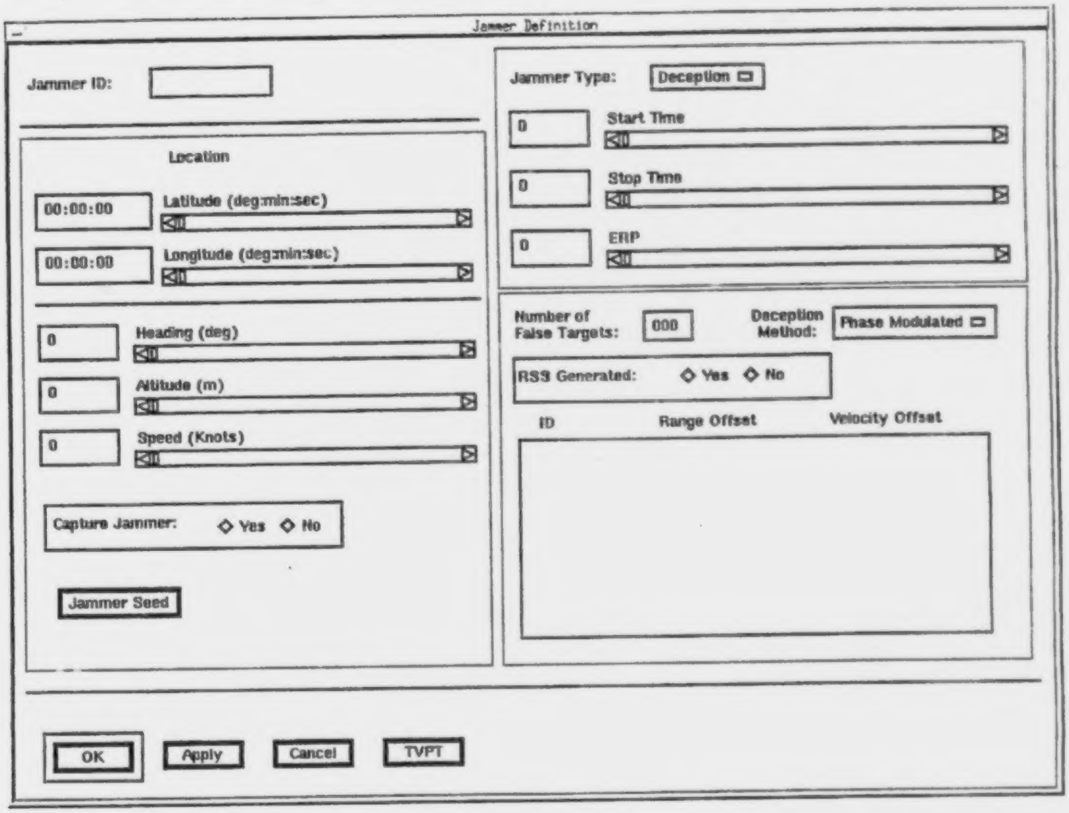
ERP: 0

Bandwidth (KHz): 0

Centre Frequency (KHz): 0

Strobe Frequency (KHz): 1.0

Duty Cycle (%): 1.0


Jammer Definition
Location
ID
Range Offset
Velocity Offset
Jammer Seed
OK
Apply
Cancel
TVPT

Jammer ID:

Latitude (deg:min:sec) 00:00:00

Longitude (deg:min:sec) 00:00:00

Heading (deg) 0

Altitude (m) 0

Speed (Knots) 0

Capture Jammer: ◇ Yes ◇ No

Jammer Type: Deception

Start Time 0

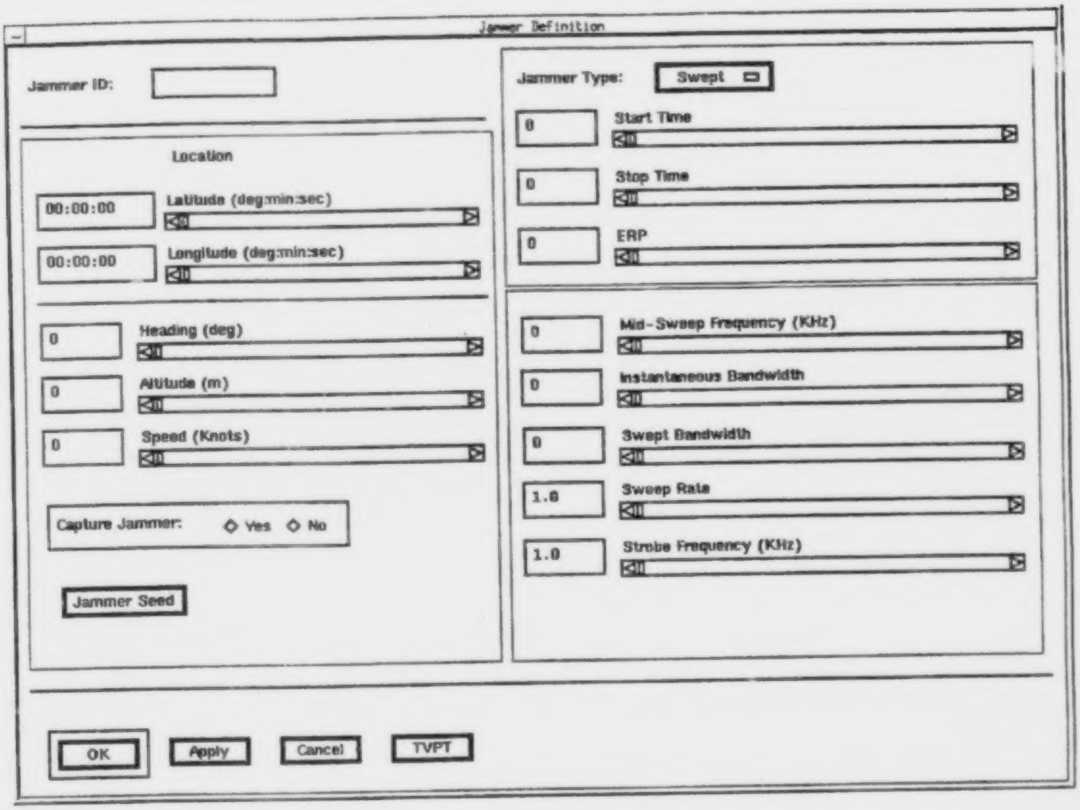
Stop Time 0

ERP 0

Number of False Targets: 000

Deception Method: Phase Modulated

RSS Generated: ◇ Yes ◇ No



Jammer Definition

Jammer ID:

Location

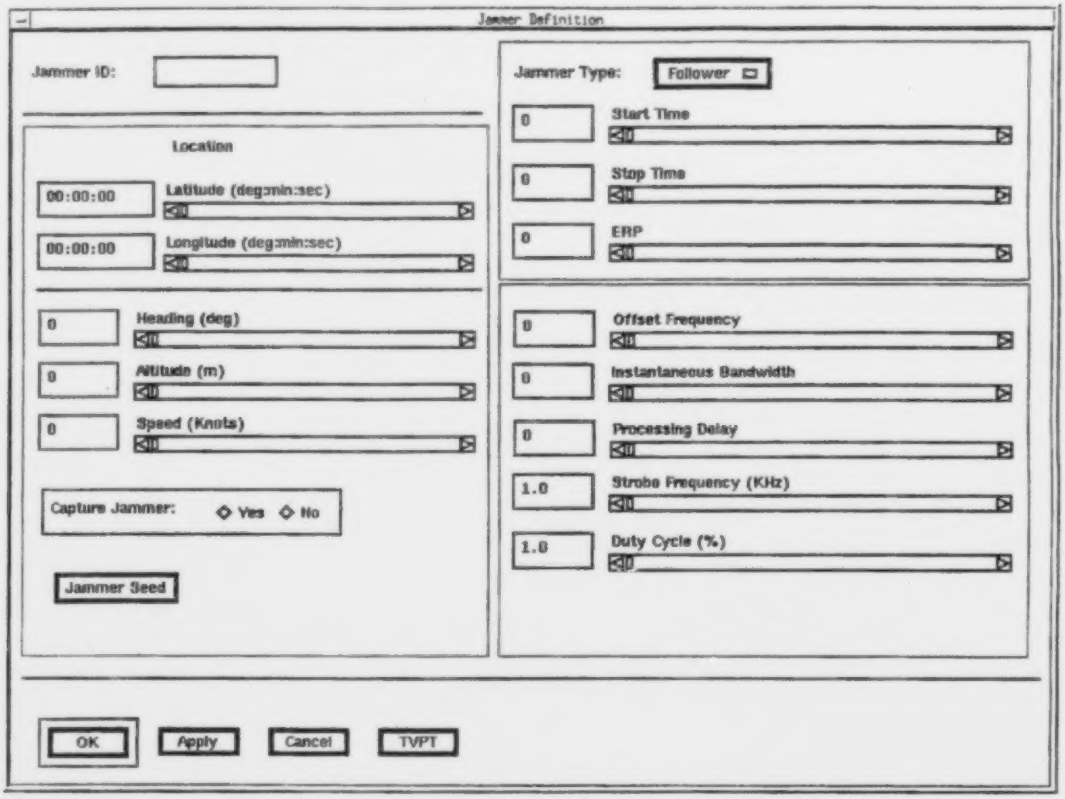
Latitude (deg:min:sec): 00:00:00
Longitude (deg:min:sec): 00:00:00
Heading (deg): 0
Altitude (m): 0
Speed (Knots): 0

Capture Jammer: ◇ Yes ◇ No

Jammer Type: Swept

Start Time: 0
Stop Time: 0
ERP: 0

Mid-Sweep Frequency (KHz): 0
Instantaneous Bandwidth: 0
Swept Bandwidth: 0
Sweep Rate: 1.0
Strobe Frequency (KHz): 1.0



Jammer Definition

Jammer ID:

Jammer Type: Follower

Location

Latitude (deg:min:sec): 00:00:00

Longitude (deg:min:sec): 00:00:00

Heading (deg): 0

Altitude (m): 0

Speed (Knots): 0

Capture Jammer: ◇ Yes ◇ No

Start Time: 0

Stop Time: 0

ERP: 0

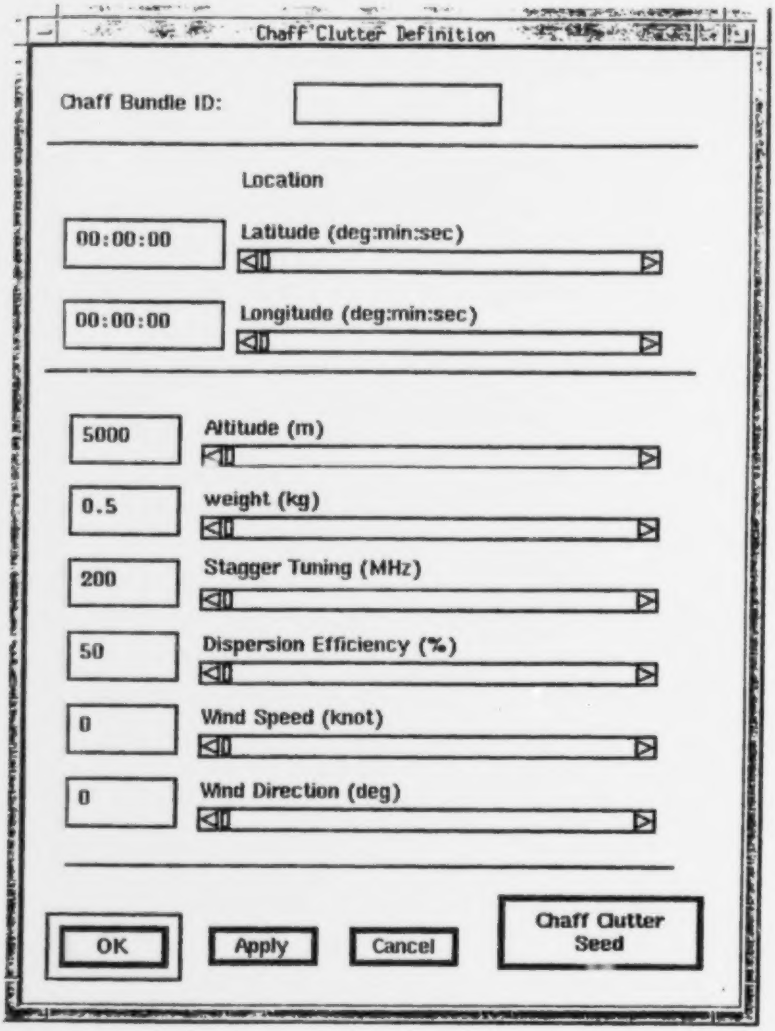
Offset Frequency: 0

Instantaneous Bandwidth: 0

Processing Delay: 0

Strobe Frequency (KHz): 1.0

Duty Cycle (%): 1.0


Chaff Clutter Definition
OK
Apply
Cancel
Chaff Clutter Seed

Chaff Bundle ID:

Location

Latitude (deg:min:sec): 00:00:00

Longitude (deg:min:sec): 00:00:00

Altitude (m): 5000

weight (kg): 0.5

Stagger Tuning (MHz): 200

Dispersion Efficiency (%): 50

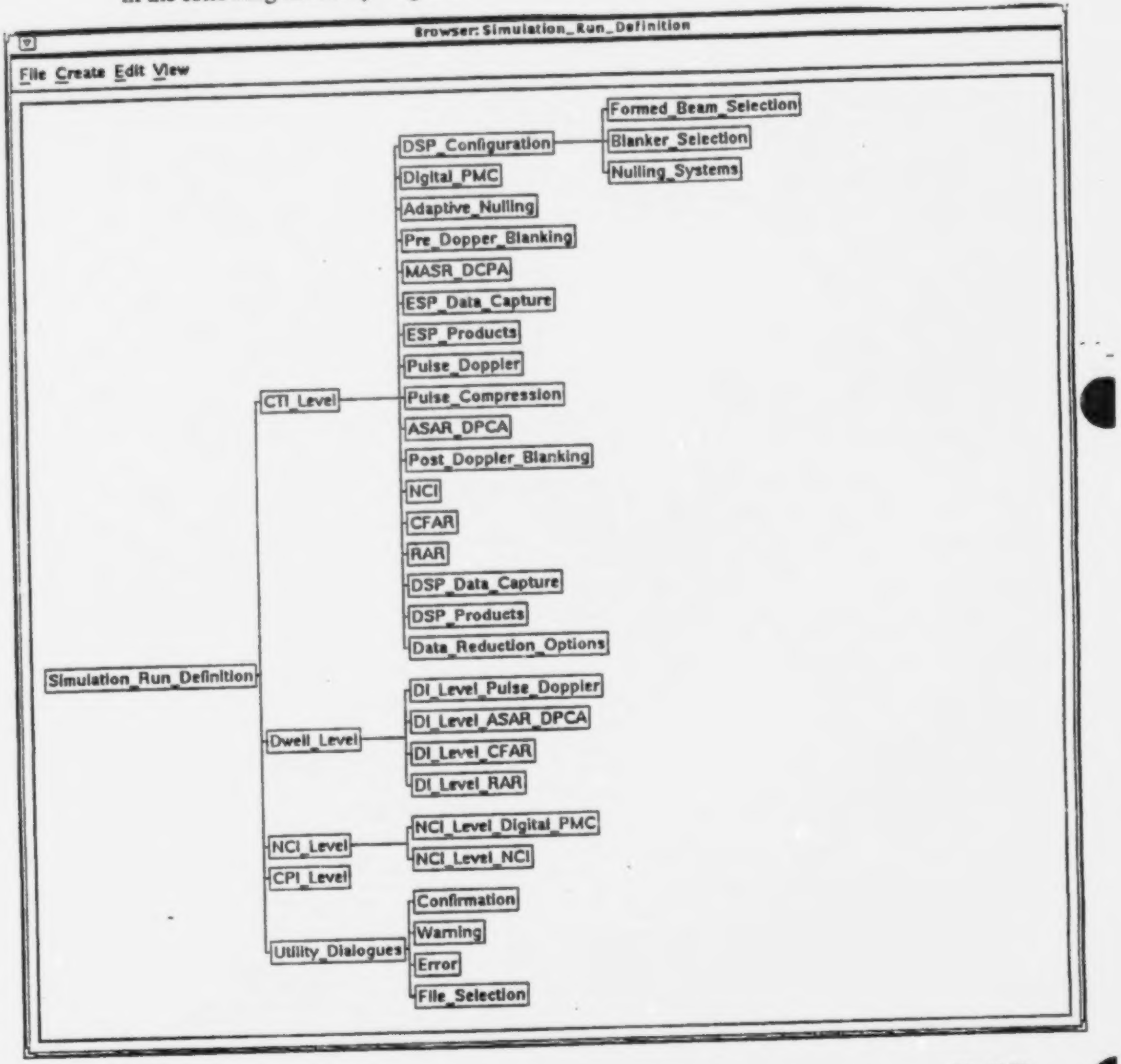
Wind Speed (knot): 0

Wind Direction (deg): 0

G4 SIMULATION RUN DEFINITION (CSDH CSU)

G4.1 - Dialogue Hierarchy Diagram

The Simulation Run Definition HMI task is implemented by utilizing a tree-like structure of dialogue boxes. Each dialogue box represents a parameter group in the definition. The hierarchy of the dialogue boxes is based on the Simulation Run Definition organization described in the Simulation Run HMI parameter Table. The dialogue hierarchy can be depicted in the following hierarchy diagram.


Browser: Simulation_Run_Definition
File
Create
Edit
View
Simulation_Run_Definition
CTI_Level
DSP_Configuration
Formed_Beam_Selection
Blanker_Selection
Nulling_Systems
Digital_PMC
Adaptive_Nulling
Pre_Dopper_Blanking
MASR_DCPA
ESP_Data_Capture
ESP_Products
Pulse_Doppler
Pulse_Compression
ASAR_DPCA
Post_Doppler_Blanking
NCI
CFAR
RAR
DSP_Data_Capture
DSP_Products
Data_Reduction_Options
Dwell_Level
DI_Level_Pulse_Doppler
DI_Level_ASAR_DPCA
DI_Level_CFAR
DI_Level_RAR
NCI_Level
NCI_Level_Digital_PMC
NCI_Level_NCI
CPI_Level
Utility_Dialogues
Confirmation
Warning
Error
File_Selection

G4.2 Top Level Dialogue (csdhTopLevelD)

The top level dialogue box of the Simulation Run Definition looks very similar to that of the Satellite/Radar Definition (Refer to Section G2.2) except that the parameter groups are further divided into two major functional areas: ESP and RSP processing.

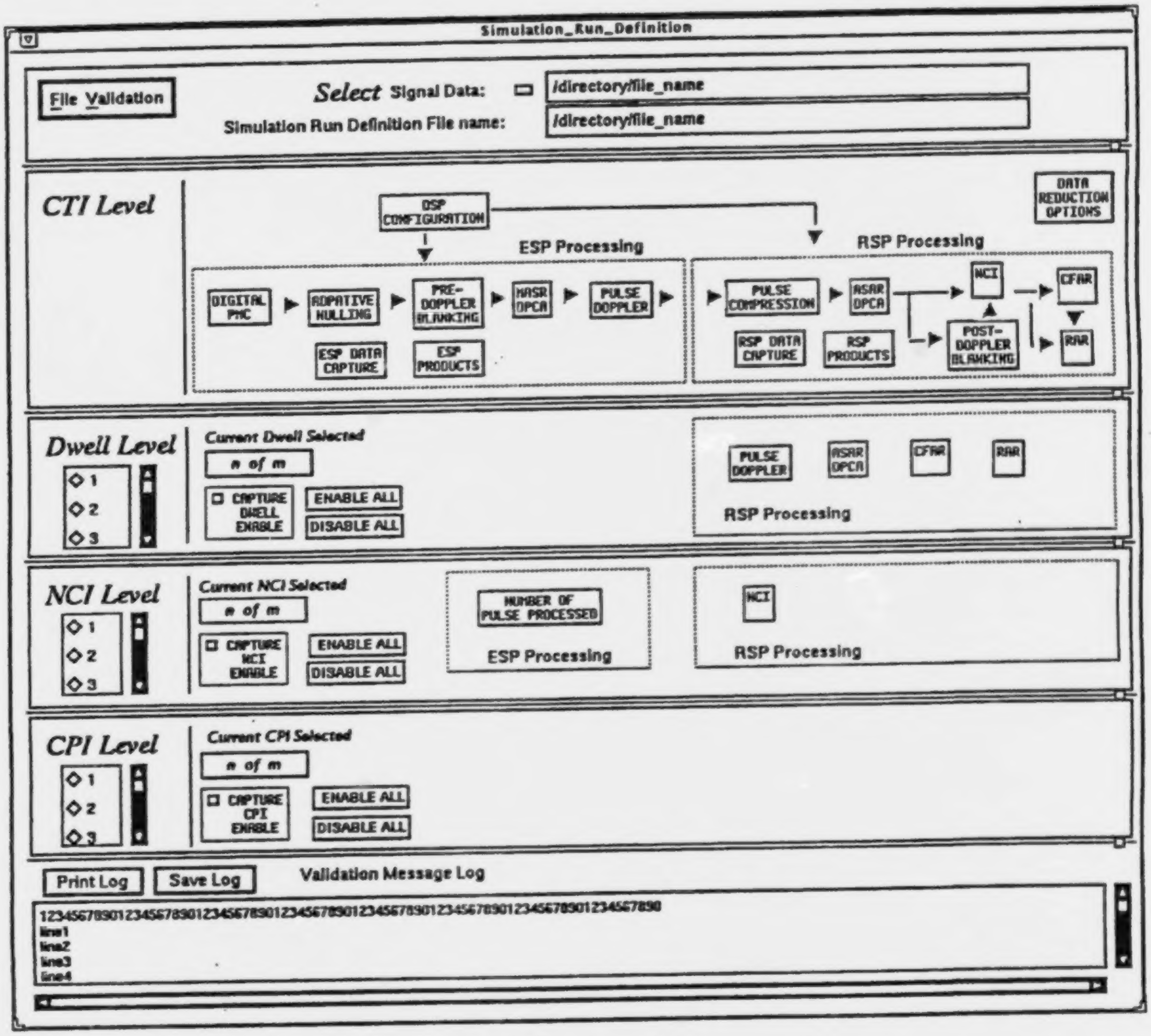
A function is activated by choosing a menu in the pull-down menu bar. The pull-down menu is described as follows:

```
pull-down menu
{                                     Actions
    File
    {
        New        Create an in-memory definition structure with defaults. The definition
                   name is defaulted to "Untitled". The user can then use the "Save"
                   function to save the new definition to the named file.
        Open       Open an existing Simulation Run Definition file. A file selection
                   dialogue appears for selecting a filename for reading the existing
                   Simulation Run Definition file into an in-memory definition structure.
        Save       Save the in-memory structure to file.
        Save As    Save the in-memory structure to the named file. A file selection
                   dialogue appears for entering a new file name.
        Exit       Prompt the user with the confirmation dialogue. If the user clicks
                   "OK", pop-down all sub-level windows if any, close and exit this
                   window. A warning dialogue will be displayed, if the user made
                   changes and has not yet performed a "Save" function.
    }
    Validate
    {
        start      Start a black-box validation using CVAL. The error messages
                   returned from the CVAL function will be displayed in the Validation
                   Error Log area.
    }
}
```

The key feature of the visualization window is that it gives the user a quick look at both "functional" and "temporal" associations. The parameters are grouped by function and each parameter group is represented as a push button on the visualization window. Depending on the changeability level, the parameter group buttons appear in either CTI, Dwell, NCI or CPI Level area. The user can select a dwell, a NCI or a CPI interval via the scrolled button list in the associated level area. The selection display indicates which dwell, NCI and CPI are current. Clicking one of these buttons invokes the "Edit" function for that parameter group and the appropriate lower level dialogues appear for parameter input.

Before activating the "New" or "Open" functions, the user must select a RSS derived processing parameters file, which is an output file from the previous signal generation run. This file can be selected from the file selection dialogue obtained by pressing the "signal data" push button. This is necessary because the Simulation Run Definition has external dependency on a subset of the Satellite/Radar Definition parameters (number of DPCA groups, number of phase centres per group, number of auxiliaries/delay taps, number of dwells, number of NCI intervals, number of CPI intervals), which are extracted from the RSS derived processing parameter file. Refer to Section A6.3.

The structure of the temporal intervals, which contains dwells, NCIs and CPIs, is constructed using the associated Satellite/Radar Definition referred by the selected RSS derived processing parameter file. Since this temporal structure is fixed (defined by the previous signal generation activity), unlike the Satellite/Radar Definition, the "Create", "Copy" and "Delete" functions are not provided by the Simulation Run Definition dialogue.


Simulation_Run_Definition
File Validation
Select
CTI Level
DSP CONFIGURATION
DATA REDUCTION OPTIONS
ESP Processing
RSP Processing
DIGITAL PMC
ADAPTIVE NULLING
PRE-DOPPLER BLANKING
MASR DPCA
PULSE DOPPLER
ESP DATA CAPTURE
ESP PRODUCTS
PULSE COMPRESSION
ASAR DPCA
NCI
CFAR
RAR
POST-DOPPLER BLANKING
RSP DATA CAPTURE
RSP PRODUCTS
Dwell Level
◇1
◇2
◇3
☐ CAPTURE DWELL ENABLE
ENABLE ALL
DISABLE ALL
PULSE DOPPLER
ASAR DPCA
CFAR
RAR
RSP Processing
NCI Level
◇1
◇2
◇3
☐ CAPTURE NCI ENABLE
ENABLE ALL
DISABLE ALL
NUMBER OF PULSE PROCESSED
ESP Processing
NCI
RSP Processing
CPI Level
◇1
◇2
◇3
☐ CAPTURE CPI ENABLE
ENABLE ALL
DISABLE ALL
Print Log
Save Log

Signal Data: /directory/file_name

Simulation Run Definition File name: /directory/file_name

Current Dwell Selected: n of m

Current NCI Selected: n of m

Current CPI Selected: n of m

Validation Message Log

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
line1
line2
line3
line4
```

csdhTopLevelD

G4.3 DSP Configuration (csdhDspConfigurationD)

This dialogue box permits the user to make changes to parameters associated with the "RSP Configuration" and "ESP Configuration" definition described in Simulation Run HMI parameter Table 1.2, 2.1.1 and 2.1.2.1.

This dialogue contains two areas:

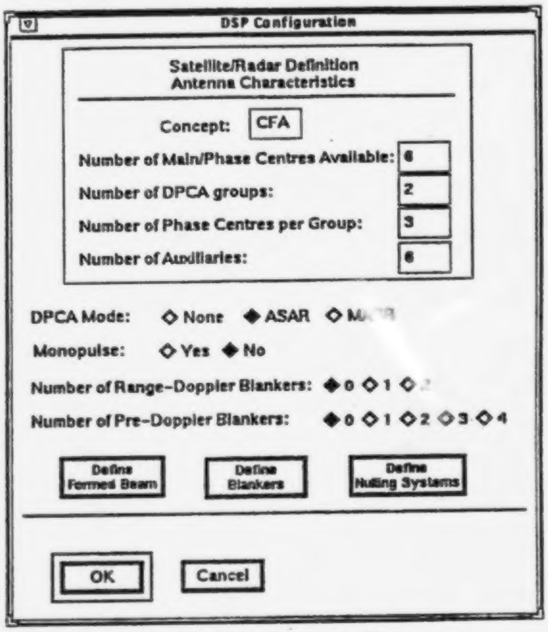
• Antenna Characteristics	Display only. Data extracted from the Satellite/Radar Definition associated with the selected RSS derived processing parameters file.
• RSP/ESP Configuration	User enters values.

Clicking the "Define Formed Beam" push button activates the Formed Beam Selection dialogue (See Section G4.4) for selecting formed beams.

Clicking the "Define Blanker" push button activates the Blanker Selection dialogue (See Section G4.5) for selecting both range-doppler blankers and pre-doppler blankers.

Clicking the "Define Nulling Systems" push button activates the Nulling Systems dialogue (See Section G4.6) for selecting nulling channels for the main channel(s) and blankers.

Smart ranges will be implemented for DPCA mode, monopulse enabled, and number of range-doppler and pre-doppler blanker fields . These parameters are dependent on the number of phase centres and auxiliaries/taps available.



Satellite/Radar Definition
Antenna Characteristics

Concept: CFA
Number of Main/Phase Centres Available: 6
Number of DPCA groups: 2
Number of Phase Centres per Group: 3
Number of Auxiliaries: 6

DPCA Mode: ◇ None ◆ ASAR ◇ MA[illegible]
Monopulse: ◇ Yes ◆ No
Number of Range-Doppler Blankers: ◆ 0 ◇ 1 ◇ [illegible]
Number of Pre-Doppler Blankers: ◆ 0 ◇ 1 ◇ 2 ◇ 3 ◇ 4

csdhDspConfigurationD

G4.4 Formed Beam Selection (csdhFormedBeamSelD)

This dialogue box permits the user to graphically select the combination of phase centres or phase centre pairs for the "Formed Beams" definition described in Simulation Run HMI parameter Table 2.1.2.2 - 2.1.2.4.

The possible choices of phase centre combination are presented in a grid button area with buttons "on" filled) or "off" indicating which phase centres are selected to make a formed beam. These buttons are displayed only; i.e. cannot be pressed. If monopulse is on, two formed beams will be displayed; otherwise one formed beam will be displayed. The choices for selection vary depending on the configuration parameters: DPCA mode, monopulse enabled, number of DPCA group and the number of phase centres per group that have been defined in the DSP Configuration Dialogue (See G4.3). The grid button display is changed corresponding to these configuration parameters. On the right side of the grid button area, the phase centres are represented with a short name considering of one of the letters "L" for leading, "M" for middle and "T" for trailing, followed by the associated channel identifier.

The user is asked to make a selection on the choices of phase centres or phase centre pairs by pressing one of the buttons in the "Select" button row appearing on the top of the dialogue. Only one choice can be made; i..e. the user can only press one button to "on" at a time. The user can make a change of selection by deselecting the current choice (toggling the "On" button to "Off") and making a new selection.

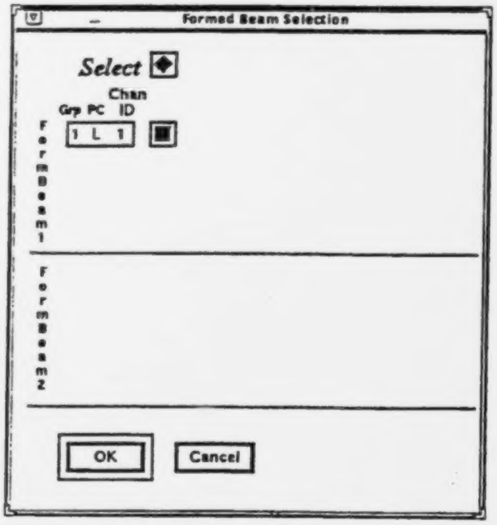
Smart ranges and defaults will be implemented for the phase centres selected to make formed beams (See PC or PC Pair Selection Table in Dependency Logic for Simulation Run HMI in Appendix A of the Testbed Specification [Document A-2]).

The formed beam selections corresponding to all possible configurations are shown in the following dialogues. The configuration description which appears at the bottom of the dialogue has the following conventions:

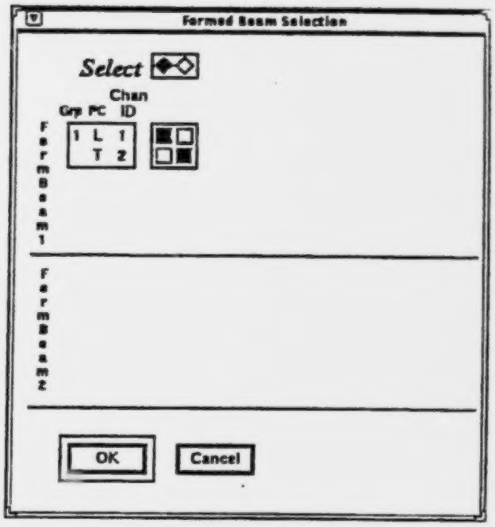
[Concept]_[DPCA Mode]_[On | Off]_m_n

where

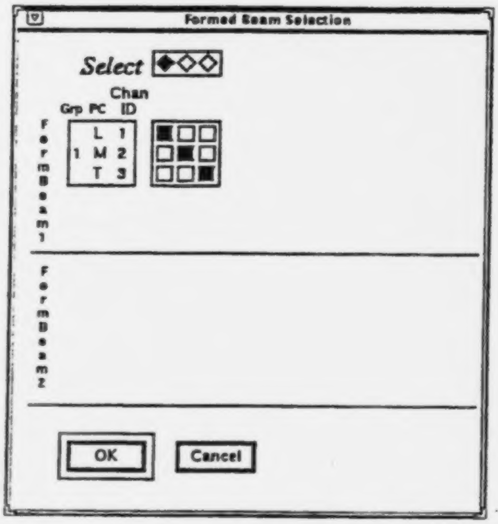
Concept is either RRA or CFA;
DPCA Mode is either None, ASAR, MASR;
On | Off indicates whether monopulse is on or not;
m: number of DPCA group;
n: number of phase centres per group.


Formed Beam Selection
Select ◆
Grp PC Chan ID
1 L 1
Form Beam 1
Form Beam 2
OK
Cancel

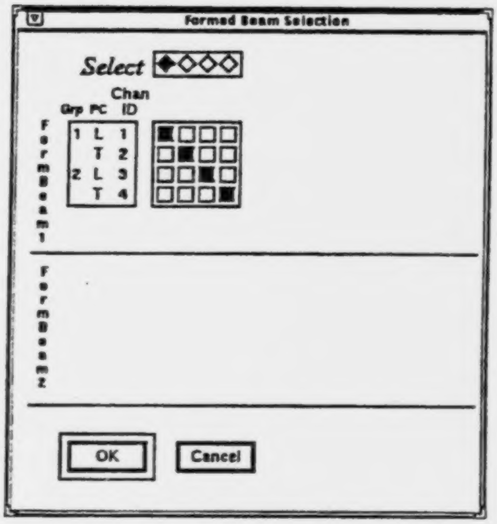
RRA_None_Off_1_1


Formed Beam Selection
Select ◆◇
Grp PC Chan ID
1 L 1
T 2
Form Beam 1
Form Beam 2
OK
Cancel

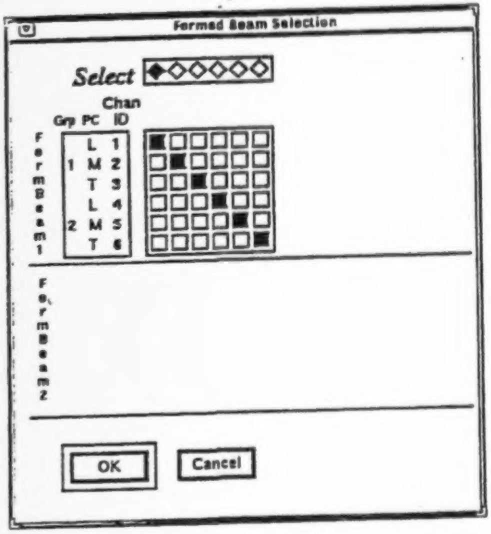
CFA_None_Off_1_2


Formed Beam Selection
Select ◆◇◇
Grp PC Chan ID
L 1
1 M 2
T 3
Form Beam 1
Form Beam 2
OK
Cancel

CFA_None_Off_1_3


Formed Beam Selection
Select ◆◇◇◇
Grp PC Chan ID
1 L 1
T 2
2 L 3
T 4
Form Beam 1
Form Beam 2
OK
Cancel

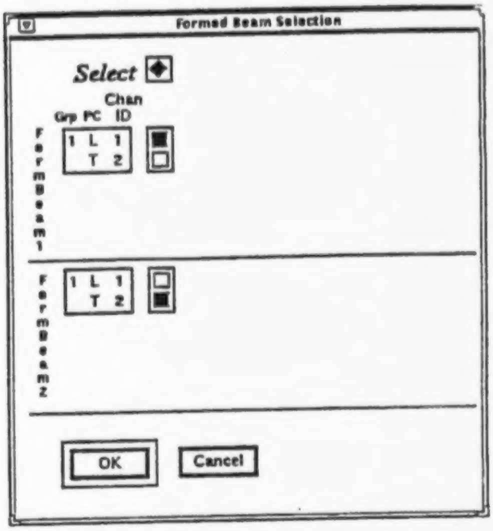
CFA_None_Off_2_2


Formed Beam Selection
Form Beam 1
Form Beam 2
OK
Cancel

Select ◆◇◇◇◇◇

Grp	PC	Chan ID						
	L	1	■	□	□	□	□	□
1	M	2	□	■	□	□	□	□
	T	3	□	□	■	□	□	□
	L	4	□	□	□	■	□	□
2	M	5	□	□	□	□	■	□
	T	6	□	□	□	□	□	■

CFA_None_Off_2_3

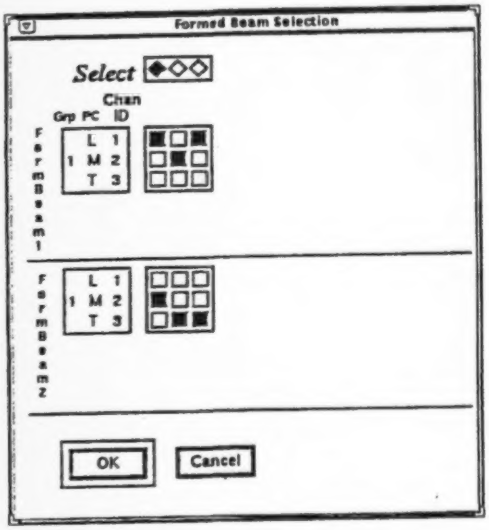

Formed Beam Selection
Form Beam 1
Form Beam 2
OK
Cancel

Select ◆

Grp	PC	Chan ID	
1	L	1	■
	T	2	□

1	L	1	□
	T	2	■

CFA_None_On_1_2

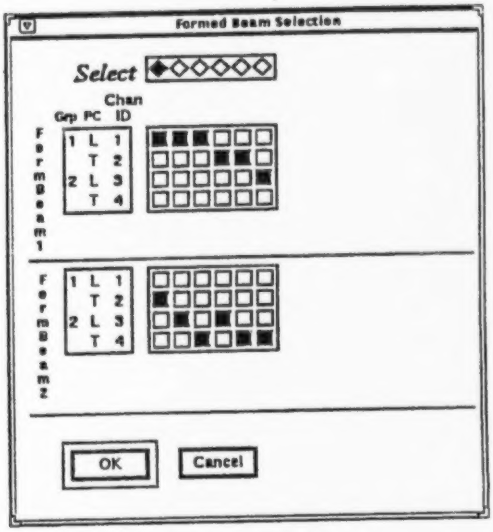

Formed Beam Selection
Form Beam 1
Form Beam 2
OK
Cancel

Select ◆◇◇

Grp	PC	Chan ID			
	L	1	■	□	■
1	M	2	□	■	□
	T	3	□	□	□

	L	1	□	□	□
1	M	2	■	□	□
	T	3	□	■	■

CFA_None_On_1_3

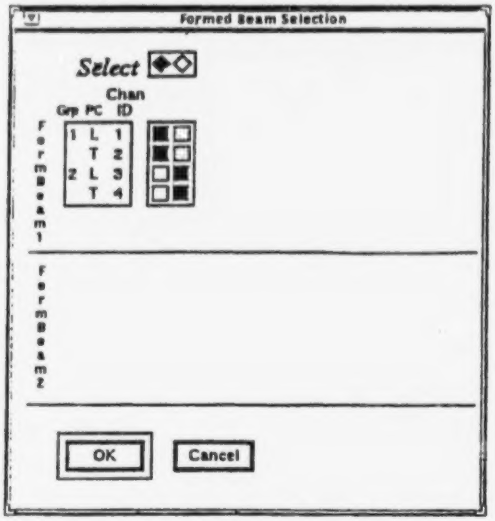

Formed Beam Selection
Form Beam 1
Form Beam 2
OK
Cancel

Select ◆◇◇◇◇◇

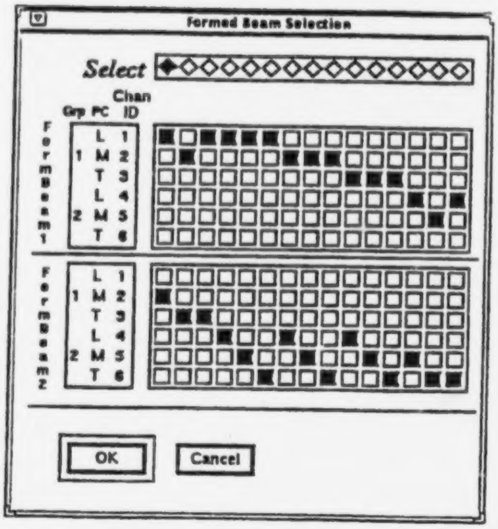
Grp	PC	Chan ID						
1	L	1	■	■	■	□	□	□
	T	2	□	□	□	■	■	□
2	L	3	□	□	□	□	□	■
	T	4	□	□	□	□	□	□

1	L	1	□	□	□	□	□	□
	T	2	■	□	□	□	□	□
2	L	3	□	■	□	■	□	□
	T	4	□	□	■	□	■	■

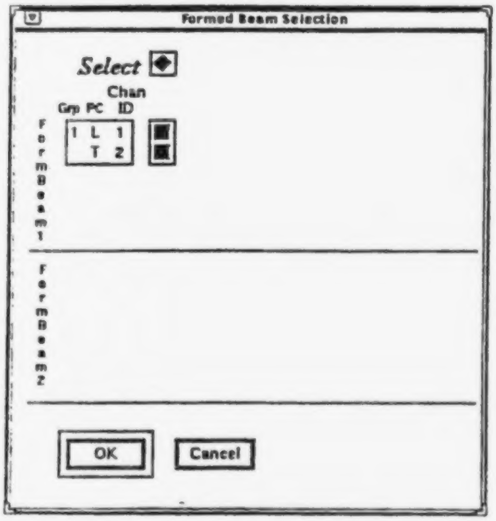
CFA_None_On_2_2


Formed Beam Selection
Select
Grp PC Chan ID
Form Beam 1
Form Beam 2
OK
Cancel

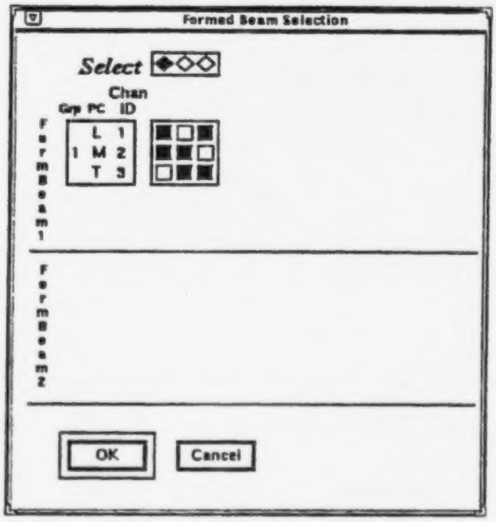
CFA_ASAR_Off_2_2


Formed Beam Selection
Select
Grp PC Chan ID
Form Beam 1
Form Beam 2
OK
Cancel

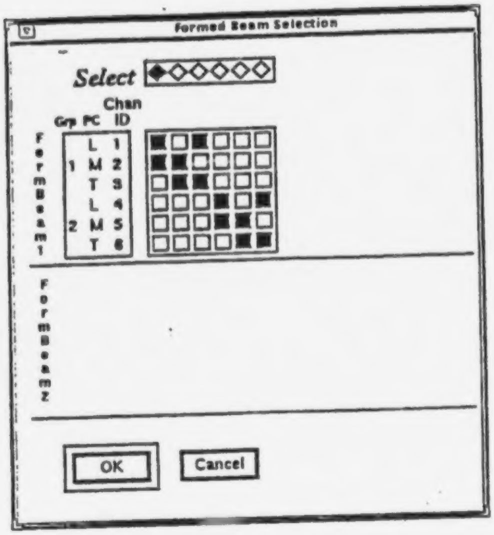
CFA_None_On_2_3


Formed Beam Selection
Select
Grp PC Chan ID
Form Beam 1
Form Beam 2
OK
Cancel

CFA_ASAR_Off_1_2


Formed Beam Selection
Select
Grp PC Chan ID
Form Beam 1
Form Beam 2
OK
Cancel

CFA_ASAR_Off_1_3

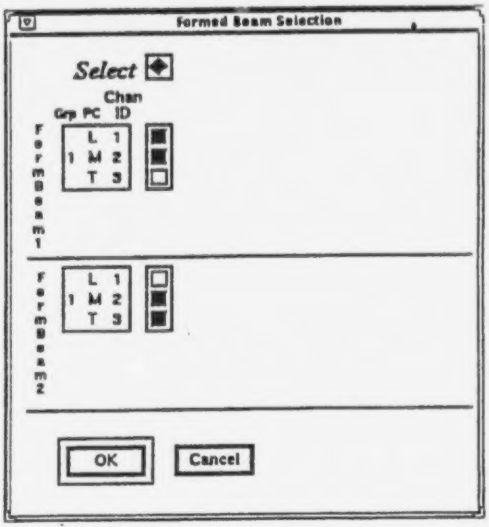

Formed Beam Selection
Select ◆◇◇◇◇◇
OK
Cancel

Form Beam 1

Grp	PC	Chan ID						
	L	1	■	☐	■	☐	☐	☐
1	M	2	■	■	☐	☐	☐	☐
	T	3	☐	■	■	☐	☐	☐
	L	4	☐	☐	☐	■	☐	■
2	M	5	☐	☐	☐	■	■	☐
	T	6	☐	☐	☐	☐	■	■

Form Beam 2

CFA_ASAR_Off_2_3


Formed Beam Selection
Select ◆
OK
Cancel

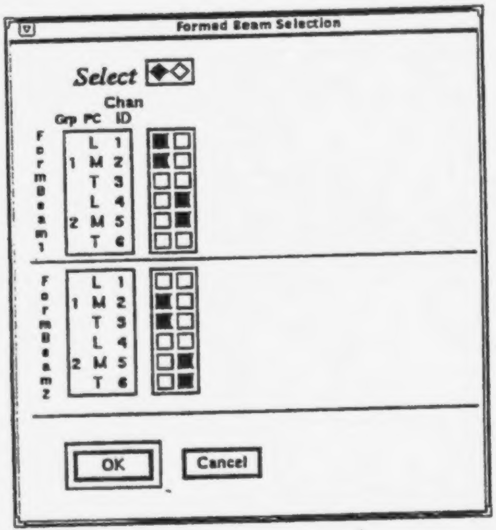
Form Beam 1

Grp	PC	Chan ID	
	L	1	■
1	M	2	■
	T	3	☐

Form Beam 2

Grp	PC	Chan ID	
	L	1	☐
1	M	2	■
	T	3	■

CFA_ASAR_On_1_3


Formed Beam Selection
Select ◆◇
OK
Cancel

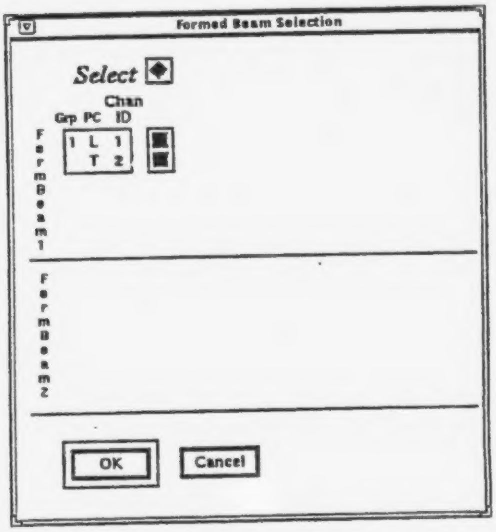
Form Beam 1

Grp	PC	Chan ID		
	L	1	■	☐
1	M	2	■	☐
	T	3	☐	☐
	L	4	☐	■
2	M	5	☐	■
	T	6	☐	☐

Form Beam 2

Grp	PC	Chan ID		
	L	1	☐	☐
1	M	2	■	☐
	T	3	■	☐
	L	4	☐	☐
2	M	5	☐	■
	T	6	☐	■

CFA_ASAR_On_2_3

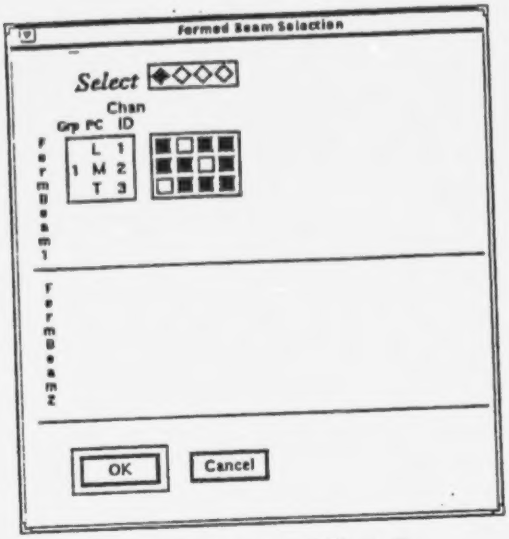

Formed Beam Selection
Select ◆
OK
Cancel

Form Beam 1

Grp	PC	Chan ID	
1	L	1	■
	T	2	■

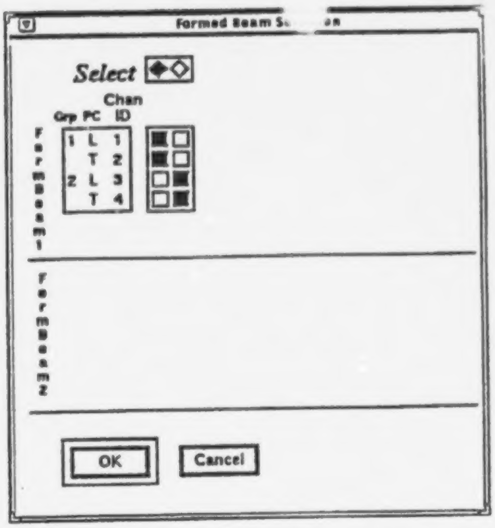
Form Beam 2

CFA_MASR_Off_1_2


Formed Beam Selection
Select ◆◇◇◇
Form Beam 1
Form Beam 2
OK
Cancel

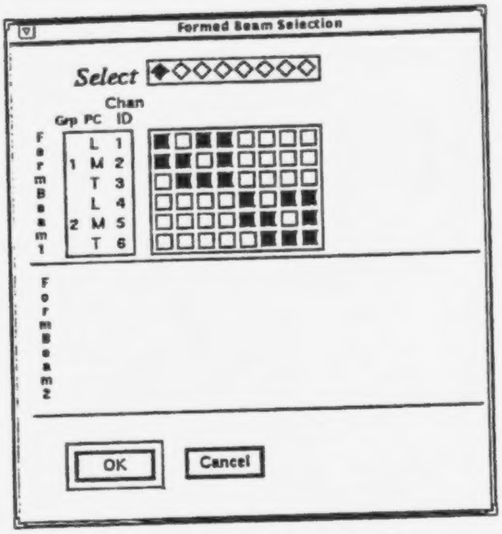
Grp	PC	Chan ID				
	L	1	■	☐	■	■
1	M	2	■	■	☐	■
	T	3	☐	■	■	■

CFA_MASR_Off_1_3


Formed Beam S... on
Select ◆◇
Form Beam 1
Form Beam 2
OK
Cancel

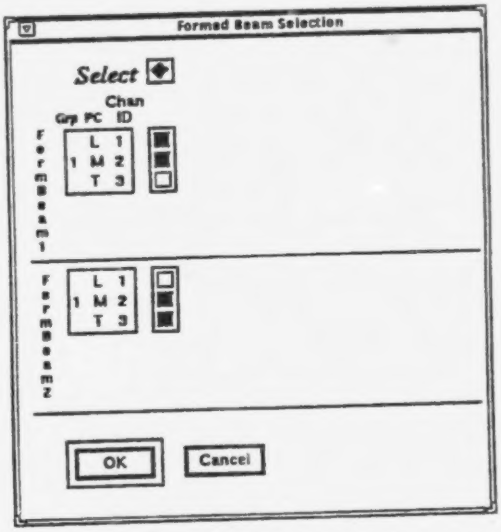
Grp	PC	Chan ID		
1	L	1	■	☐
	T	2	■	☐
2	L	3	☐	■
	T	4	☐	■

CFA_MASR_Off_2_2


Formed Beam Selection
Select ◆◇◇◇◇◇◇◇
Form Beam 1
Form Beam 2
OK
Cancel

Grp	PC	Chan ID								
	L	1	■	☐	■	■	☐	☐	☐	☐
1	M	2	■	■	☐	■	☐	☐	☐	☐
	T	3	☐	■	■	■	☐	☐	☐	☐
	L	4	☐	☐	☐	☐	■	☐	■	■
2	M	5	☐	☐	☐	☐	■	■	☐	■
	T	6	☐	☐	☐	☐	☐	■	■	■

CFA_MASR_Off_2_3


Formed Beam Selection
Select ◆
Form Beam 1
Form Beam 2
OK
Cancel

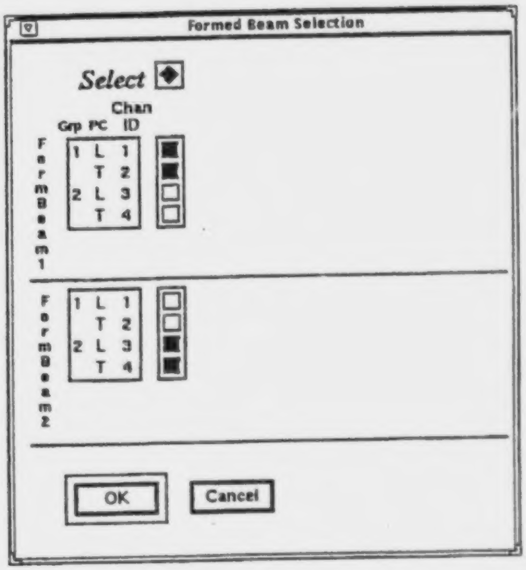
Form Beam 1

Grp	PC	Chan ID	
	L	1	■
1	M	2	■
	T	3	☐

Form Beam 2

Grp	PC	Chan ID	
	L	1	☐
1	M	2	■
	T	3	■

CFA_MASR_On_1_3


Formed Beam Selection
Select ◆
OK
Cancel

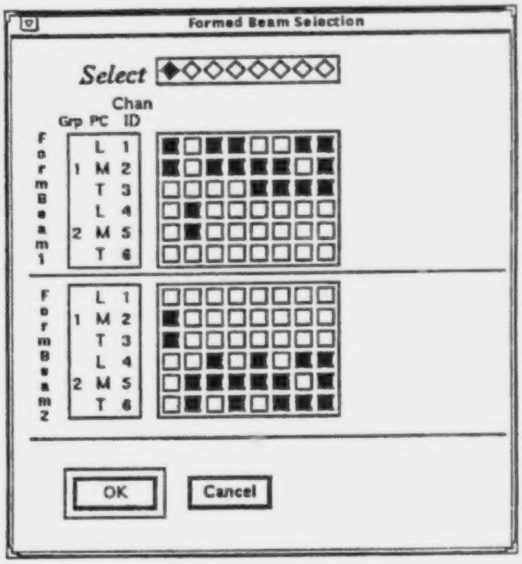
Form Beam 1

Grp	PC	Chan ID	
1	L	1	■
	T	2	■
2	L	3	☐
	T	4	☐

Form Beam 2

Grp	PC	Chan ID	
1	L	1	☐
	T	2	☐
2	L	3	■
	T	4	■

CFA_MASR_On_2_2


Formed Beam Selection
Select ◆◇◇◇◇◇◇◇
OK
Cancel

Form Beam 1

Grp	PC	Chan ID								
	L	1	■	☐	■	■	☐	☐	■	■
1	M	2	■	☐	■	■	■	■	☐	■
	T	3	☐	☐	☐	☐	■	■	■	■
	L	4	☐	■	☐	☐	☐	☐	☐	☐
2	M	5	☐	■	☐	☐	☐	☐	☐	☐
	T	6	☐	☐	☐	☐	☐	☐	☐	☐

Form Beam 2

Grp	PC	Chan ID								
	L	1	☐	☐	☐	☐	☐	☐	☐	☐
1	M	2	■	☐	☐	☐	☐	☐	☐	☐
	T	3	■	☐	☐	☐	☐	☐	☐	☐
	L	4	☐	☐	■	☐	■	☐	■	■
2	M	5	☐	■	■	■	■	■	☐	■
	T	6	☐	■	☐	■	☐	■	■	■

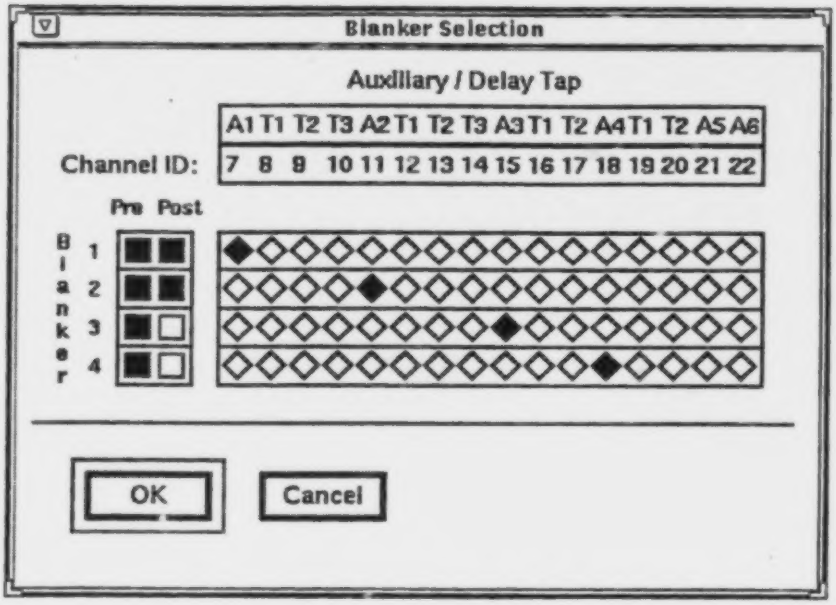
CFA_MASR_On_2_3

G4.5 Blanker Selection (csdhBlankerSelD)

This dialogue box permits the user to graphically select range-doppler blankers and pre-doppler blankers for the "Blankers" definition described in Simulation Run HMI parameter Table 1.2.3, 2.1.4 and 2.1.4.1.

Both auxiliary/delay tap acronyms and channel IDs are displayed. The channel ID is derived using the channel assignment logic described in the Dependency Logic for Simulation Run HMI of the Testbed Specification (Document A-2). The user chooses auxiliaries or delay taps for blankers (range-doppler or pre-doppler blankers) by turning on radio buttons on the grid button area; Only available auxiliaries/delay taps are sensitive for selection.

The square buttons display the number of pre-Doppler blankers (Pre) and the number of Range-Doppler blankers previously entered.



			A1	T1	T2	T3	A2	T1	T2	T3	A3	T1	T2	A4	T1	T2	A5	A6
Channel ID:			7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Blanker	Pre	Post																
1	■	■	◆	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇
2	■	■	◇	◇	◇	◇	◆	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇
3	■	☐	◇	◇	◇	◇	◇	◇	◇	◇	◆	◇	◇	◇	◇	◇	◇	◇
4	■	☐	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇	◆	◇	◇	◇	◇

csdhBlankerSelD

G4.6 Nulling Systems (csdhNullingSystemsD)

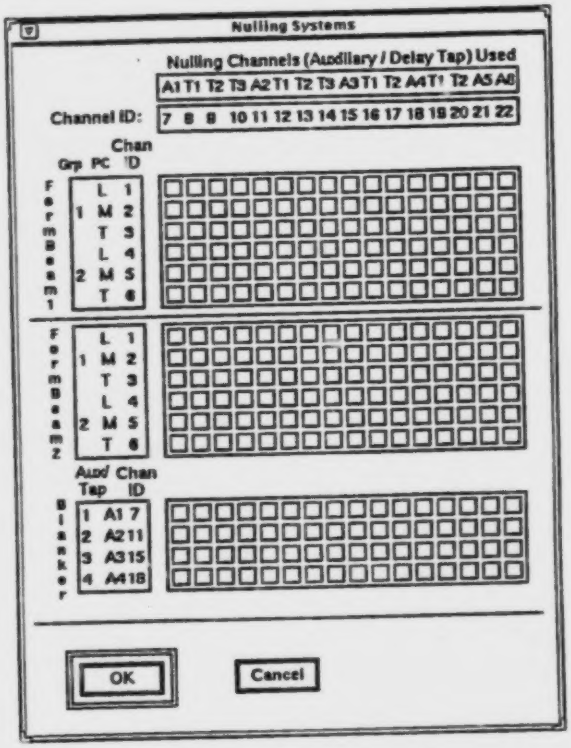
This dialogue box permits the user to graphically select nulling channels for the "Nulling Systems" definition described in Simulation Run HMI parameter Table 2.1.3.

This dialogue contains two grid button areas (arrays of square buttons):

Formed beams	Nulling channels for Main channel/phase centres
Blankers	Nulling channels for pre-doppler and range doppler blankers

Each square button may be used to select one of the auxiliary antennas or delay taps. Both auxiliary/delay tap acronyms and channel IDs are displayed. The channel ID is derived using the channel assignment logic described in the Dependency Logic for Simulation Run HMI of the Testbed Specification (Document A-2).

Smart ranges and smart defaults will be implemented for nulling auxiliaries and tap delays.



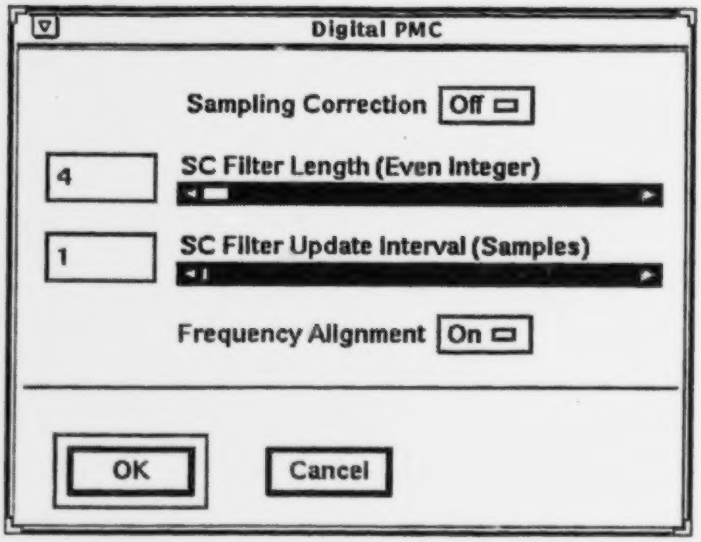
Nulling Channels (Auxiliary / Delay Tap) Used
A1 T1 T2 T3 A2 T1 T2 T3 A3 T1 T2 A4 T1 T2 A5 A6
Channel ID: 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

Nulling Systems
Grp PC Chan ID
Form Beam 1
1 L 1
M 2
T 3
2 L 4
M 5
T 6
Form Beam 2
1 L 1
M 2
T 3
2 L 4
M 5
T 6
Blanker
Aux/ Tap Chan ID
1 A1 7
2 A2 11
3 A3 15
4 A4 18
OK
Cancel

csdhNullingSystemsD

G4.7 Digital PMC (csdhDigitalPmcD)

This dialogue box permits the user to make changes to those parameters associated with the "Digital PMC" definition which are changeable at CTI level, as described in Simulation Run HMI parameter Table 2.2.4.



Sampling Correction: Off

SC Filter Length (Even Integer): 4

SC Filter Update Interval (Samples): 1

Frequency Alignment: On

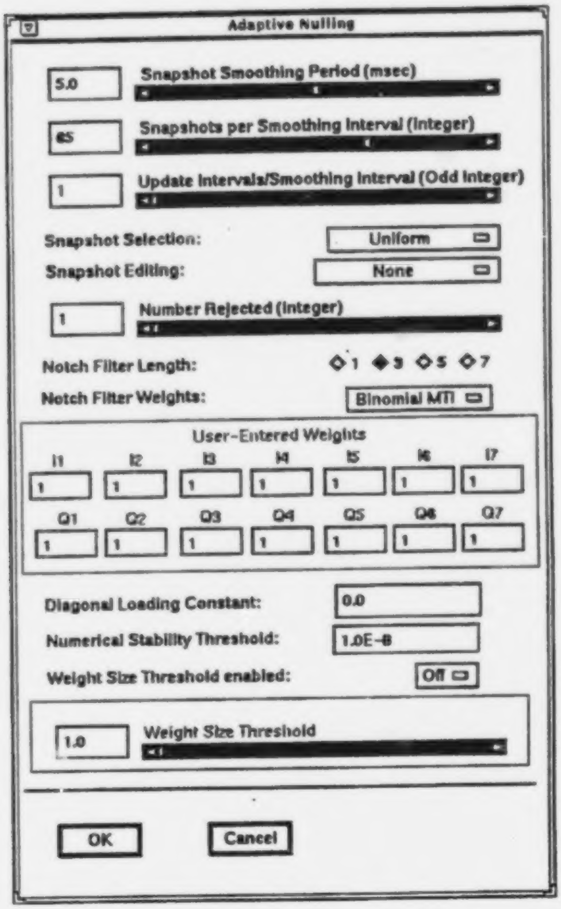
csdhDigitalPmcD

G4.8 Adaptive Nulling (csdhAdaptiveNullingD)

This dialogue box permits the user to make changes to those parameters associated with the Normal Equations" definition which are changeable at CTI level, as described in Simulation Run HMI parameter Table 2.2.1.

Fixed ranges will be implemented for the following parameters:

0.1 <= snapshot smoothing period <= 10 ns
1 <= number of snapshots per smoothing interval <= 100
1 <= update intervals/smoothing interval <= 99 (odd integer)
1 <= number rejected <= 99



Snapshot Smoothing Period (msec): 5.0
Snapshots per Smoothing Interval (Integer): 65
Update Intervals/Smoothing Interval (Odd Integer): 1
Snapshot Selection: Uniform
Snapshot Editing: None
Number Rejected (Integer): 1
Notch Filter Length: ◇ 1 ◆ 3 ◇ 5 ◇ 7
Notch Filter Weights: Binomial MTI

User-Entered Weights

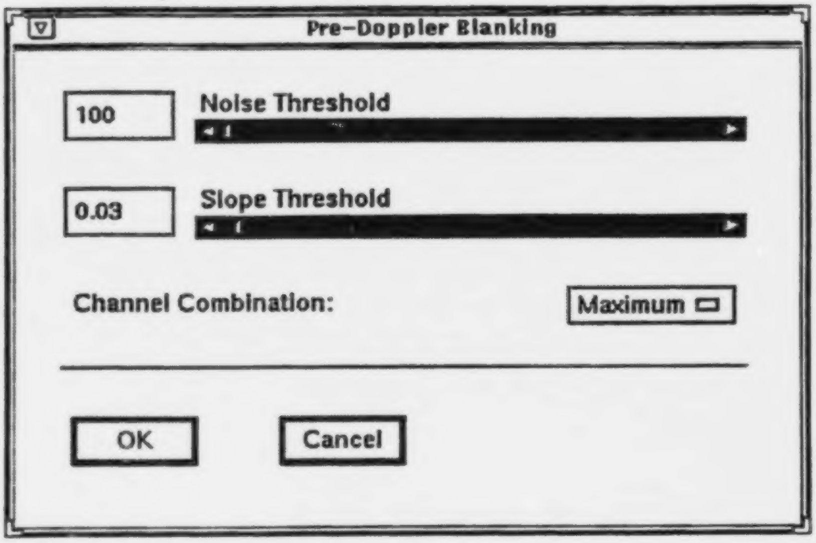
I1	I2	I3	I4	I5	I6	I7
1	1	1	1	1	1	1
Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**
1	1	1	1	1	1	1

Diagonal Loading Constant: 0.0
Numerical Stability Threshold: 1.0E-8
Weight Size Threshold enabled: Off
Weight Size Threshold: 1.0

csdhAdaptiveNullingD

G4.9 Pre-Doppler Blanking (csdhPreDopplerBlankingD)

This dialogue box permits the user to make changes to those parameters associated with the "Pre-Doppler Blanking" definition which are changeable at CTI level, as described in Simulation Run HMI parameter Table 2.2.2.



Noise Threshold: 100

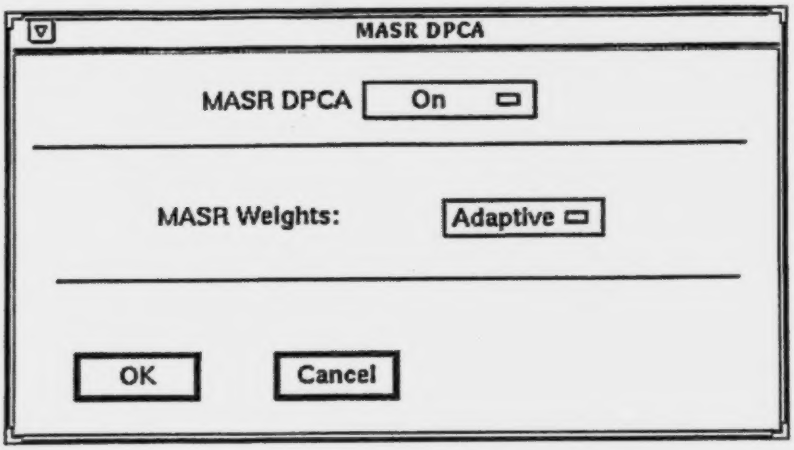
Slope Threshold: 0.03

Channel Combination: Maximum

csdhPreDopplerBlankingD

G4.10 MASR DPCA (csdhMasrDpcaD)

This dialogue box permits the user to make changes to those parameters associated with the "MASR DPCA" definition which are changeable at CTI level, as described in Simulation Run HMI parameter Table 2.2.3. This dialogue is available only if MASR is selected for DPCA mode in the DSP Configuration dialogue (See G4.3).



MASR DPCA

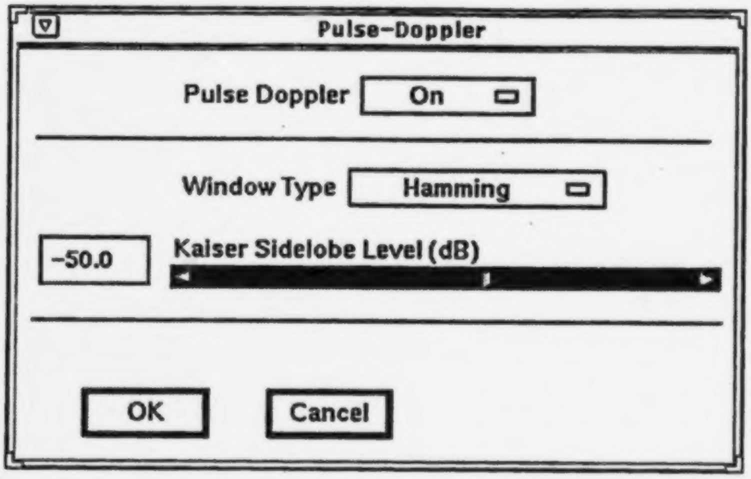
MASR DPCA: On

MASR Weights: Adaptive

csdhMasrDpcaD

G4.11 Pulse-Doppler (csdhPulseDopplerD)

This dialogue box permits the user to make changes to those parameters associated with the Pulse-Doppler" definition which are changeable at CTI level, as described in Simulation Run HMI parameter Table 2.2.5.



Pulse Doppler: On

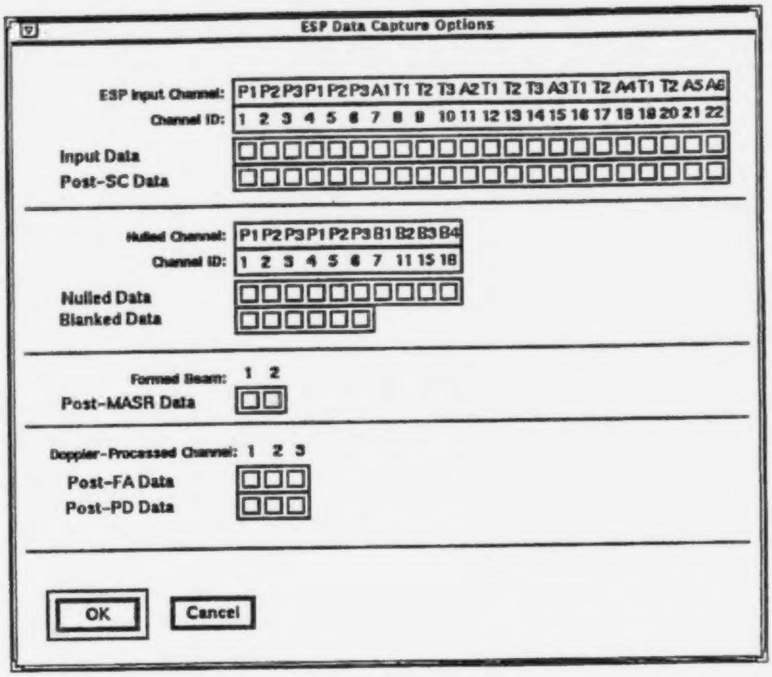
Window Type: Hamming

Kaiser Sidelobe Level (dB): -50.0

csdhPulseDopplerD

G4.12 ESP Data Capture Options (csdhEspDataCaptureD)

This dialogue box permits the user to choose the capture options for ESP signal data in the "Data Capture Options" definition in Simulation Run HMI parameter Table 2.3.1.



ESP Data Capture Options

ESP Input Channel:	P1	P2	P3	P1	P2	P3	A1	T1	T2	T3	A2	T1	T2	T3	A3	T1	T2	A4	T1	T2	A5	A6
Channel ID:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Input Data	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Post-SC Data	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Nulled Channel:	P1	P2	P3	P1	P2	P3	B1	B2	B3	B4
Channel ID:	1	2	3	4	5	6	7	11	15	18
Nulled Data	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Blanked Data	☐	☐	☐	☐	☐	☐				

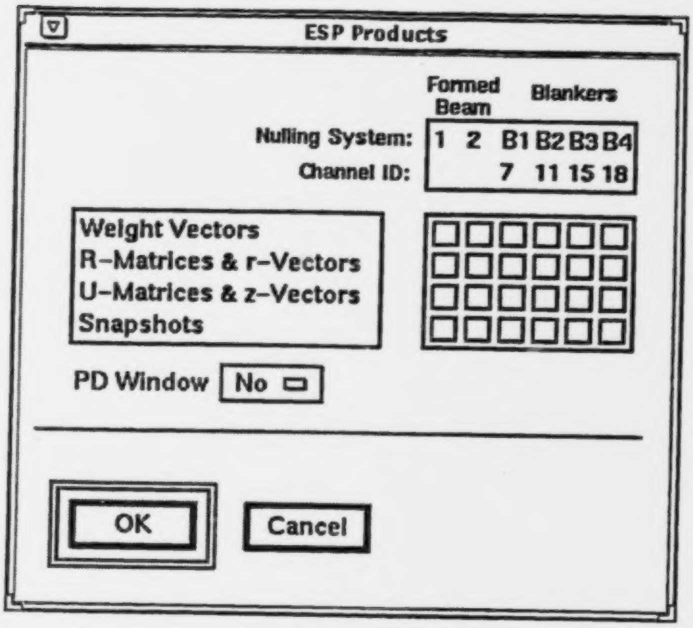
Formed Beam:	1	2
Post-MASR Data	☐	☐

Doppler-Processed Channel:	1	2	3
Post-FA Data	☐	☐	☐
Post-PD Data	☐	☐	☐

csdhEspDataCaptureD

G4.13 ESP Products (csdhEspProductsD)

This dialogue box permits the user to choose the capture options for ESP Products in the "Normal Equations Products" definition in Simulation Run HMI parameter Table 2.3.2.



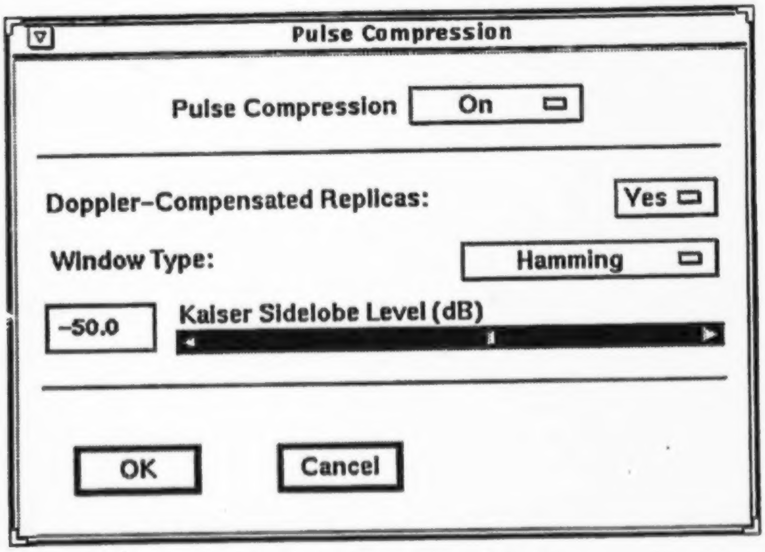
	Formed Beam		Blankers			
Nulling System:	1	2	B1	B2	B3	B4
Channel ID:			7	11	15	18
Weight Vectors	☐	☐	☐	☐	☐	☐
R-Matrices & r-Vectors	☐	☐	☐	☐	☐	☐
U-Matrices & z-Vectors	☐	☐	☐	☐	☐	☐
Snapshots	☐	☐	☐	☐	☐	☐

PD Window No

csdhEspProductsD

G4.14 Pulse Compression (csdhPulseCompressionD)

This dialogue box permits the user to make changes to those parameters associated with the "Pulse Compression" definition which are changeable at CTI level, as described in Simulation Run HMI parameter Table 2.2.6.



Pulse Compression: On

Doppler-Compensated Replicas: Yes

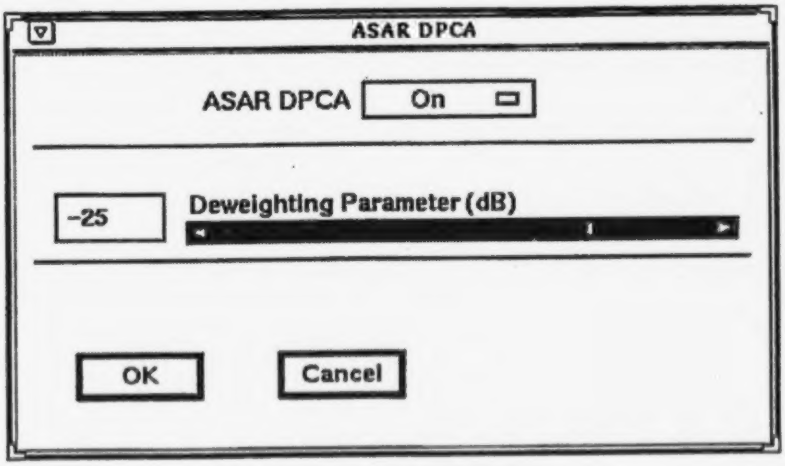
Window Type: Hamming

-50.0 Kaiser Sidelobe Level (dB)

csdhPulseCompressionD

G4.15 ASAR DPCA (csdhAsarDpcaD)

This dialogue box permits the user to make changes to parameters associated with the "ASAR DPCA" definition changeable at CTI level described in Simulation Run HMI parameter Table 2.2.7.



ASAR DPCA

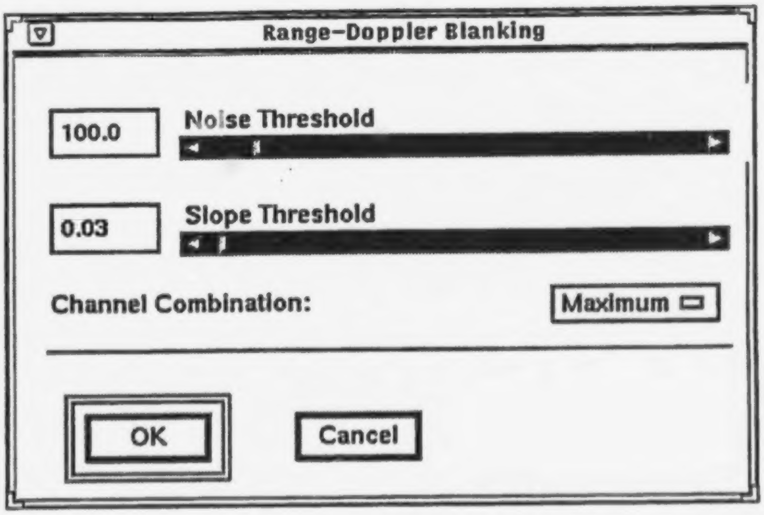
ASAR DPCA: On

-25 Deweighting Parameter (dB)

csdhAsarDpcaD

G4.16 Range-Doppler Blanking (csdhRngDopplerBlankingD)

This dialogue box permits the user to make changes to those parameters associated with the "Range-Doppler Blanking" definition which are changeable at CTI level, as described in Simulation Run HMI parameter Table 2.2.8.



Noise Threshold: 100.0

Slope Threshold: 0.03

Channel Combination: Maximum

csdhRngDopplerBlankingD

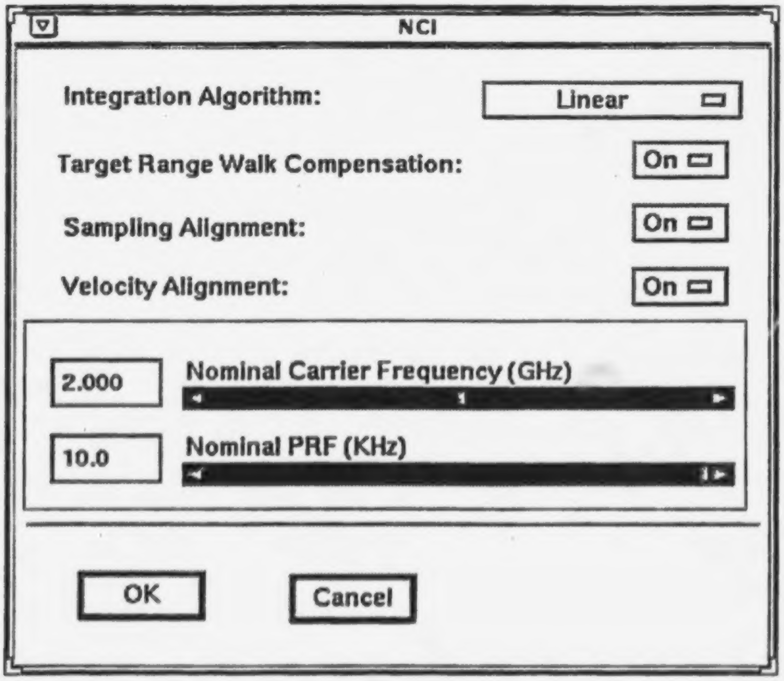
G4.17 NCI (csdhNciD)

This dialogue box permits the user to make changes to those parameters associated with the "NCI" definition which are changeable at CTI level, as described in Simulation Run HMI parameter Table 2.2.9.

Fixed ranges will be implemented for the following parameters:
1 <= nominal carrier frequency <= 4 GHz

Fixed defaults will be implemented for the following parameters:

nominal carrier frequency	2 GHz
nominal PRF	10 KHz

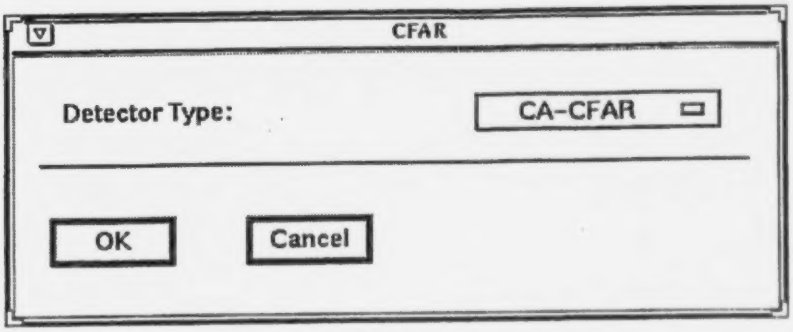


Integration Algorithm: Linear
Target Range Walk Compensation: On
Sampling Alignment: On
Velocity Alignment: On
Nominal Carrier Frequency (GHz): 2.000
Nominal PRF (KHz): 10.0

csdhNciD

G4.18 CFAR (csdhCfarD)

This dialogue box permits the user to make changes to those parameters associated with the "Detection" definition which are changeable at CTI level, as described in Simulation Run HMI parameter Table 2.2.10.



CFAR

Detector Type: CA-CFAR

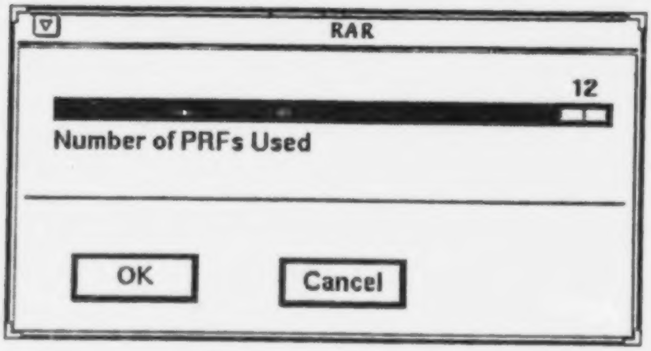
csdhCfarD

G4.19 RAR (csdhRarD)

This dialogue box permits the user to make changes to those parameters associated with the "RAR" definition which are changeable at CTI level, as described in Simulation Run HMI parameter Table 2.2.11.

Fixed ranges will be implemented for the following parameters:
1 <= number of PRFs used <= 12

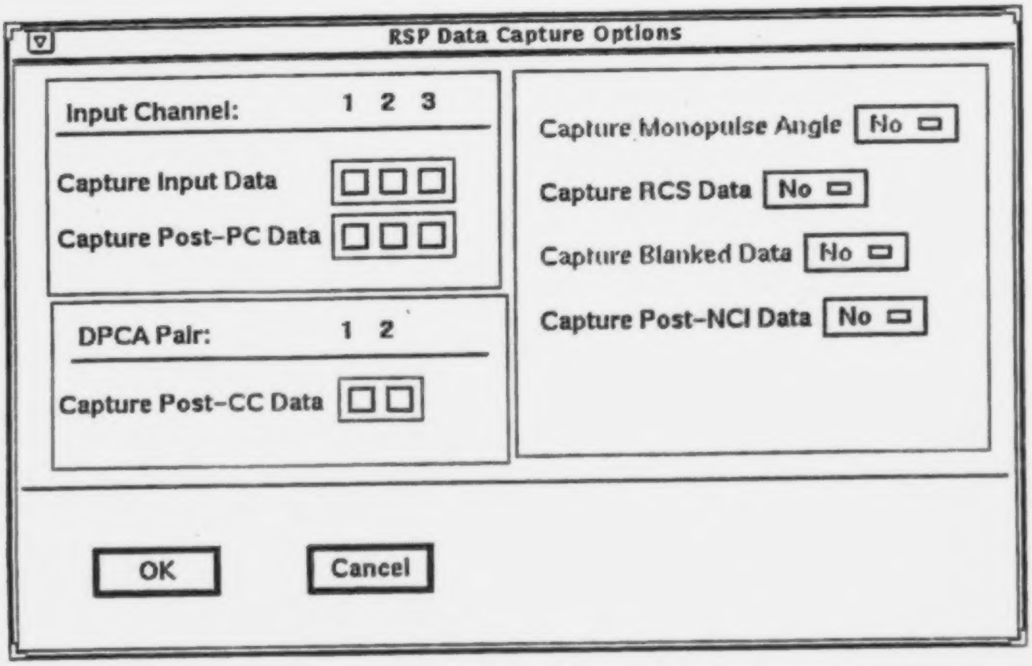
Fixed defaults will be implemented for the following parameters:
number of PRFs used 3



csdhRarD

G4.20 RSP Data Capture Options (csdhRspDataCaptureD)

This dialogue box permits the user to choose the capture options for RSP signal data in the "Data Capture Options" definition in Simulation Run HMI parameter Table 2.3.2.



Input Channel: 1 2 3

Capture Input Data ☐ ☐ ☐

Capture Post-PC Data ☐ ☐ ☐

DPCA Pair: 1 2

Capture Post-CC Data ☐ ☐

Capture Monopulse Angle No

Capture RCS Data No

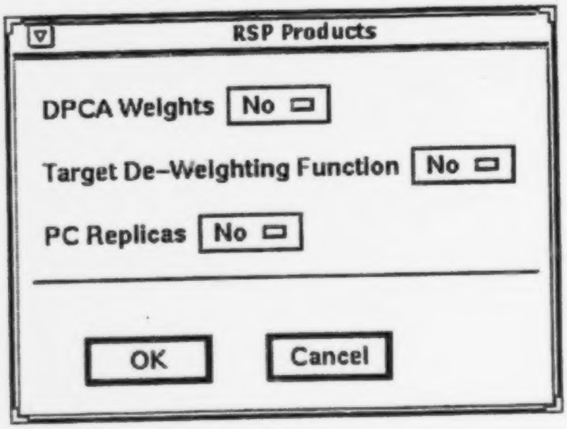
Capture Blanked Data No

Capture Post-NCI Data No

csdhRspDataCaptureD

G4.21 RSP Products (csdhRspProductsD)

This dialogue box permits the user to choose the capture options for RSP products in the "DSP Products" definition in Simulation Run HMI parameter Table 2.3.5.



RSP Products

DPCA Weights: No

Target De-Weighting Function: No

PC Replicas: No

csdhRspProductsD

G4.22 Data Reduction Options (csdhDataReductionD)

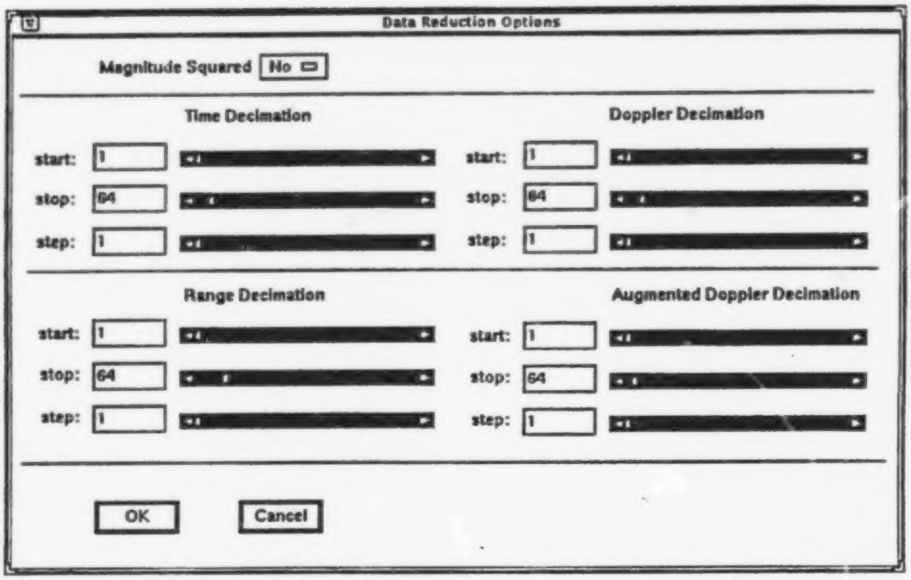
This dialogue box permits the user to choose the data reduction options for all products in the "Signal Data Reduction" definition in Simulation Run HMI parameter Table 2.3.3.

Fixed ranges will be implemented for the following parameters:

1 <= time decimation <= 1024
1 <= range decimation <= 512
1 <= doppler decimation <= 1024
1 <= augmented doppler decimation <= 2048

Fixed defaults will be implemented for the following parameters:

time decimation	1,64,1
range decimation	1,64,1
doppler decimation	1,64,1
augmented doppler decimation	1,64,1



Data Reduction Options

Magnitude Squared: No

Time Decimation
start: 1
stop: 64
step: 1

Doppler Decimation
start: 1
stop: 64
step: 1

Range Decimation
start: 1
stop: 64
step: 1

Augmented Doppler Decimation
start: 1
stop: 64
step: 1

csdhDataReductionD

G4.23 Dwell Level Pulse Doppler (csdhDiPulseDopplerD)

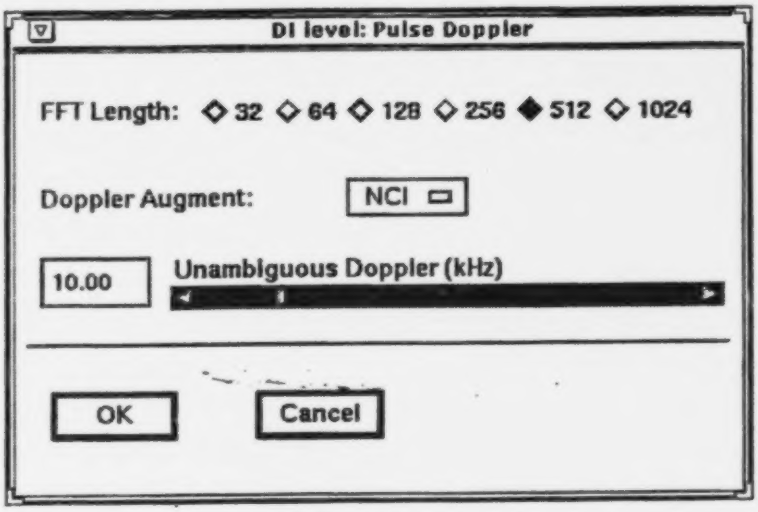
This dialogue box permits the user to make changes to those parameters associated with the "Pulse Doppler" definition which are changeable at dwell level, as described in Simulation Run HMI parameter Table 2.2.5.

Fixed ranges will be implemented for the following parameters:

32 <= FFT length <= 1024 (in powers of 2)
0.2 <= unambiguous Doppler <= 200 KHz

Fixed defaults will be implemented for the following parameters:

FFT length	RRA: 128; CFA: 512
Doppler Augment Select	NCI
unambiguous Doppler	RRA: 20 KHz; CFA: 10 KHz



Dl level: Pulse Doppler

FFT Length: ◇ 32 ◇ 64 ◇ 128 ◇ 256 ◆ 512 ◇ 1024

Doppler Augment: NCI

10.00 Unambiguous Doppler (kHz)

csdhDiPulseDopplerD

G4.24 Dwell Level ASAR DPCA (csdhDiAsarDpcaD)

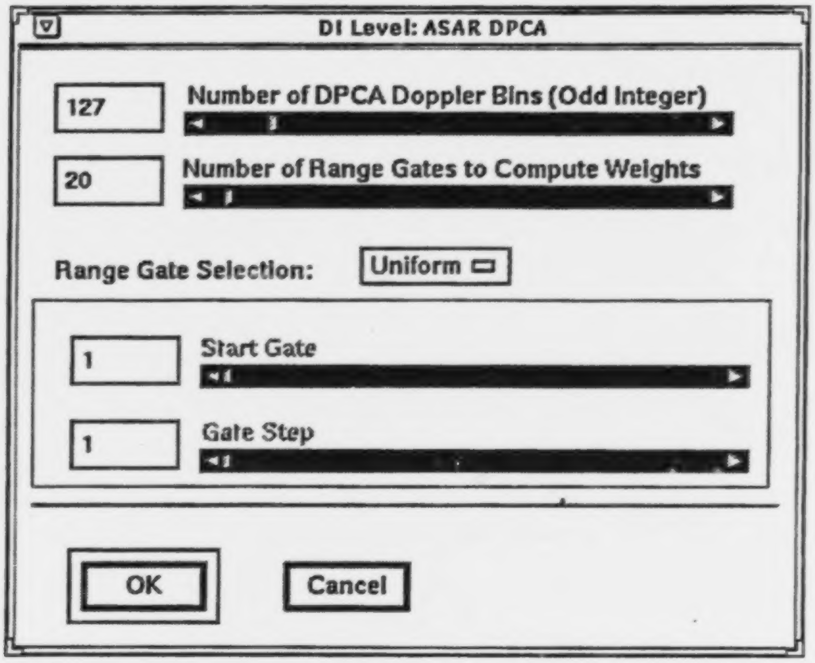
This dialogue box permits the user to make changes to those parameters associated with the "ASAR DPCA" definition which are changeable at dwell level, as described in Simulation Run HMI parameter Table 2.2.7.

Fixed ranges will be implemented for the following parameters:

1 <= number of DPCA doppler bins <= 1023 (odd integer)
1 <= number of range gated to compute weights <= 512
1 <= start gate <= 512
1 <= step < = 511

Fixed defaults will be implemented for the following parameters:

number of DPCA doppler bins	127
number of range gates used to compute weights	20



Dl Level: ASAR DPCA

Number of DPCA Doppler Bins (Odd Integer): 127

Number of Range Gates to Compute Weights: 20

Range Gate Selection: Uniform

Start Gate: 1

Gate Step: 1

csdhDiAsarDpcaD

G4.25 Dwell Level CFAR (csdhDiCfarD)

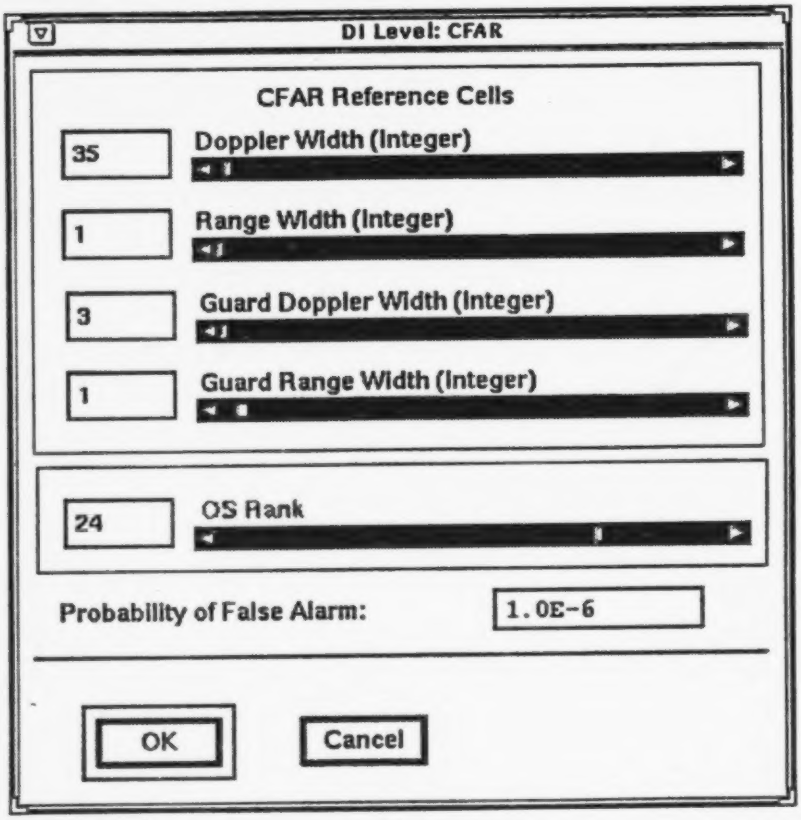
This dialogue box permits the user to make changes to those parameters associated with the "Detection" definition which are changeable at dwell level, as described in Simulation Run HMI parameter Table 2.2.10.

Fixed ranges will be implemented for the following parameters:
1 <= doppler width <= 2048
1 <= range width <= 512
0 <= guard doppler width <= 2048
0 <= guard range width <= 512

Smart range will be implemented for the following parameter:
1<=OS rank <= N

where N = Doppler Width * Range Width - Guard Doppler Width * Guard Range Width

Smart default will be implemented for the following parameter:
OS rank 3N/4



DI Level: CFAR

CFAR Reference Cells

Doppler Width (Integer): 35
Range Width (Integer): 1
Guard Doppler Width (Integer): 3
Guard Range Width (Integer): 1

OS Rank: 24

Probability of False Alarm: 1.0E-6

csdhDiCfarD

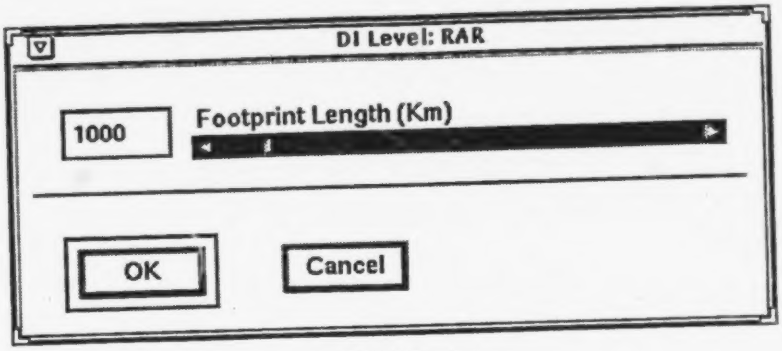
G4.26 **Dwell Level RAR (csdhDiRarD)**

This dialogue box permits the user to make changes to parameters associated with the "RAR" definition changeable at dwell level described in Simulation Run HMI parameter Table 2.2.11.

Fixed ranges will be implemented for the following parameters:
1 <= footprint length <= 10,000 Km

Fixed default will be implemented for the following parameters:

footprint length	1000 km



Footprint Length (Km): 1000

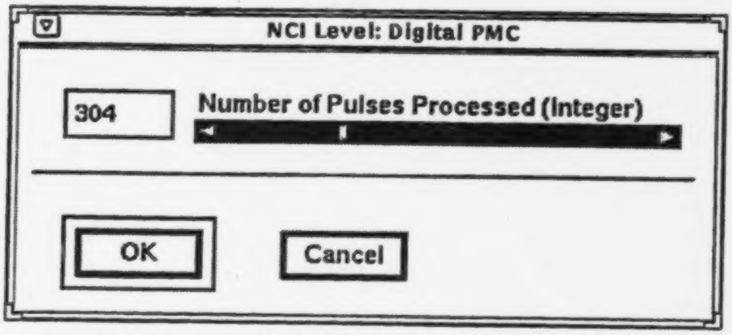
csdhDiRarD

G4.27 NCI Level Digital PMC (csdhNciDigitalPmcD)

This dialogue box permits the user to make changes to those parameters associated with the "Digital PMC" definition which are changeable at NCI interval level, as described in Simulation Run HMI parameter Table 2.2.4.

Smart range will be implemented for the following parameter:
32 <= number of pulses processed <= min. (1024, Number of Transmitted Pulses)

Smart default will be implemented for the following parameters:
number of pulses processed <= min. (1024, Number of Transmitted Pulses)



Number of Pulses Processed (Integer): 304

csdhNciDigitalPmcD

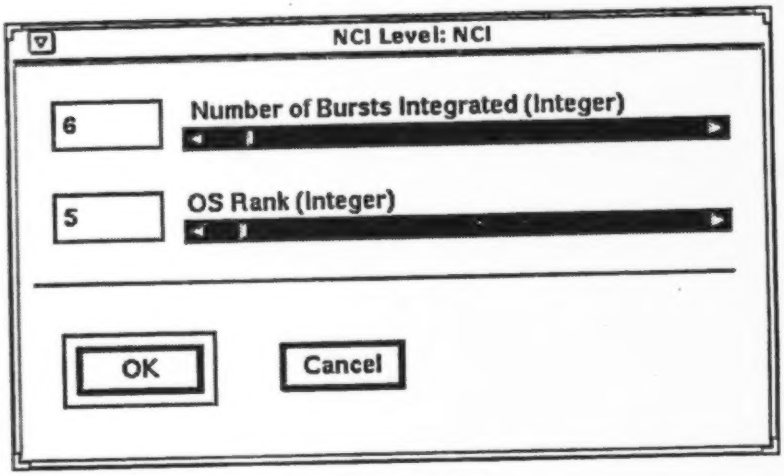
G4.28 NCI Level NCI (csdhNciNciD)

This dialogue box permits the user to make changes to those parameters associated with the "NCI" definition which are changeable at NCI interval level, as described in Simulation Run HMI parameter Table 2.2.9.

Smart range will be implemented for the following parameters:
1 <= Number of Bursts Integrated <= Number of CPIs Transmitted
1 <= OS Rank <= Number of Bursts Integrated

Smart defaults will be implemented for the following parameters:

Number of Bursts Integrated	Number of CPIs Transmitted
OS rank	Number of Bursts Integrated -1



Number of Bursts Integrated (Integer): 6
OS Rank (Integer): 5

csdhNciNciD

G5 MONITOR/CONTROL SIGNAL GENERATION (CGEH CSU)

G5.1 Top Level Dialogue (cgehTopLevelD)

This is the top level dialogue box of the Monitor/Control Signal Generation task. This top level dialogue has two main areas:

- An execution definition area
- A Validation Error Log area

Each area is designed as a pained window. The user can adjust the size of an area by scrolling the sash up or down; the sash appears as a square box positioned on the bottom of the area. For example, the user can enlarge the Validation Error Log area to display more lines of error messages.

The user must select a Satellite/Radar Definition file name and an Environment Definition file name through the file selection dialogue obtained by pressing the small push button positioned at the left side of the file name text field. (See File Selection Dialogue in G1.3)

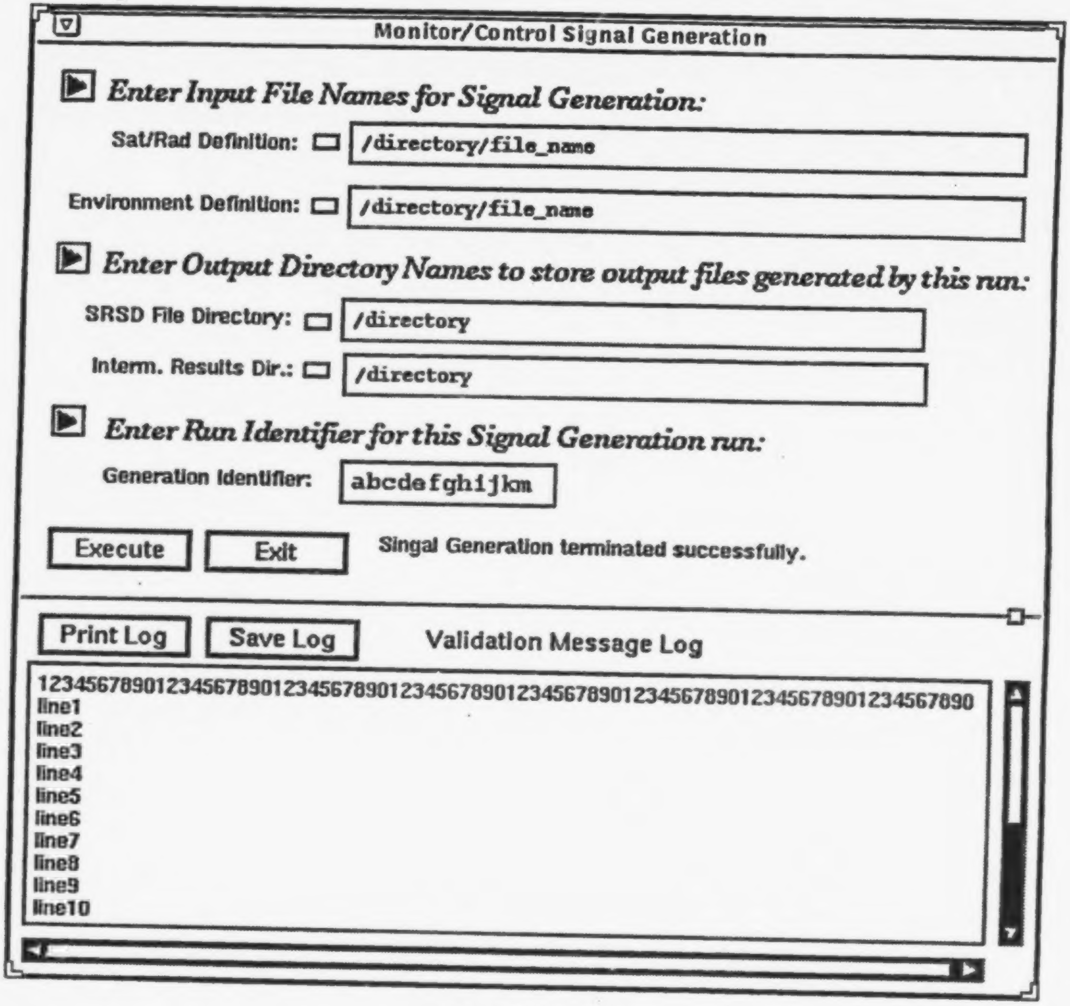
When the user clicks on the "Execute" button, the system will automatically invoke a validation check. If any error is detected, the validation error messages will be displayed in the Validation Error Log area. The user can optionally press the "Print Log" button to print or press the "Save Log" button to save the validation error log to a file. If no error was found, a signal generation activity is initiated.

If error was detected, the user can either re-select the definition file names (Simulation Run Definition and/or RSS derived processing parameters file) or proceed with erroneous parameters by pressing "Execute" button again. In the later case, a warning message will be displayed for confirmation.

If the "Exit" button is pressed, the system prompts the user with the confirmation dialogue. If the user clicks "OK", this window will be closed and this task terminates.

A message line is displayed on the Top Level Dialogue, after the user presses the "Execute" button. The message line indicates the application status and may be one of the following messages:

- "Generation run in progress, please wait..."
- "Generation request is denied"
- "Generation run terminated successfully"
- "Generation run terminated with errors"
- "Generation run aborted successfully"
- "Generation run aborted with errors"


Monitor/Control Signal Generation
Execute
Exit
Print Log
Save Log
Validation Message Log

▶ *Enter Input File Names for Signal Generation:*

Sat/Rad Definition: ☐ /directory/file_name

Environment Definition: ☐ /directory/file_name

▶ *Enter Output Directory Names to store output files generated by this run:*

SRSD File Directory: ☐ /directory

Interm. Results Dir.: ☐ /directory

▶ *Enter Run Identifier for this Signal Generation run:*

Generation Identifier: abcdefghijkm

Singal Generation terminated successfully.

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
line1
line2
line3
line4
line5
line6
line7
line8
line9
line10
```

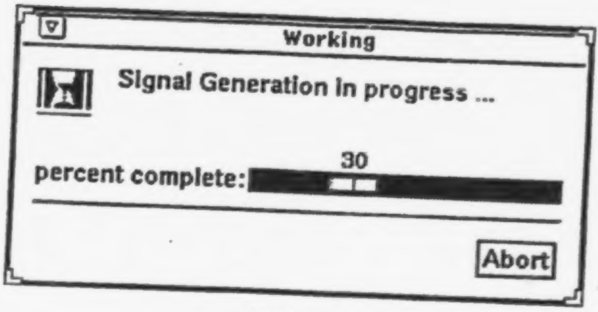
cgehTopLevelD

G5.2 Progress Status Dialogue

This dialogue is displayed while the Signal Generation activity is in progress. The scale widget indicates the percentage completion of the activity (0% to 100%). Clicking the "Abort" button will terminate the current signal generation activity and return control back to the Top Level Dialogue of Monitor/Control Signal Generation task (See G5.1).

When "Abort" is chosen, a confirmation dialogue box will be presented before Signal Generation is aborted.

While this dialogue is displayed, the top level dialogue input is blocked, and all buttons and fields in the top level dialogue are disabled.



Signal Generation in progress ...

percent complete: 30

cgehWorking

G6 MONITOR/CONTROL SIMULATION RUN (CSIH CSU)

G6.1 Top Level Dialogue (csihTopLevelD)

This is the top level dialogue box of the Monitor/Control Simulation Run task. This top level dialogue has two main areas:

- An execution definition area
- A Validation Error Log area

Each area is designed as a pained window. The user can adjust the size of an area by scrolling the sash up or down; the sash appears as a square box positioned on the bottom of the area. For example, the user can enlarge the Validation Error Log area to display more lines of error messages.

The user must select both the Simulation Run Definition file name and the RSS derived processing parameters file name through the file selection dialogue obtained by pressing the small push button positioned at the left side of the file name text field. (See File Selection Dialogue in G1.3)

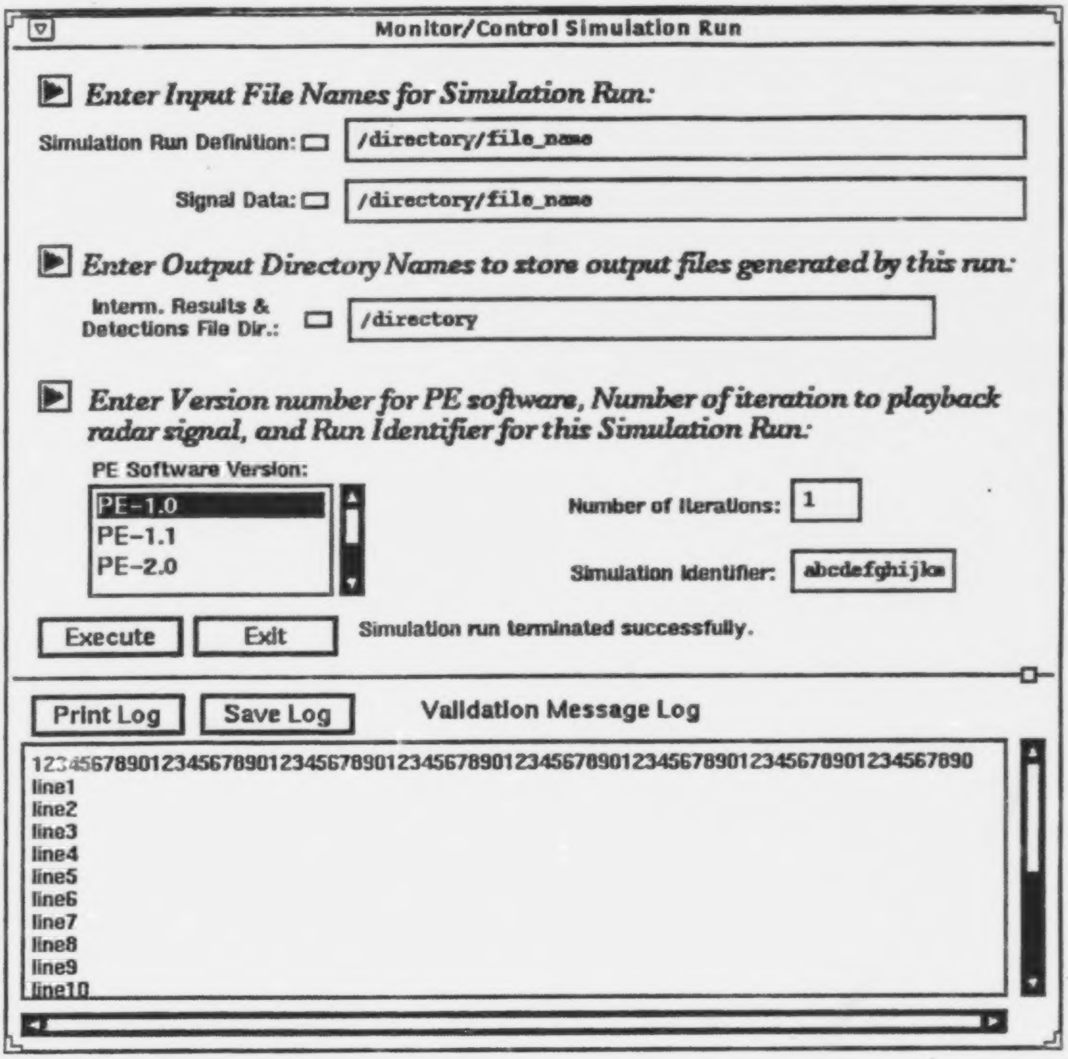
When the user clicks on the "Execute" button, the system will automatically invoke a validation check. If any error is detected, the validation error messages will be displayed in the Validation Error Log area. The user can optionally press the "Print Log" button to print or press the "Save Log" button to save the validation error log to a file. If no error was found, a simulation run activity is initiated.

If error was detected, the user can either re-select the definition names (Simulation Run Definition and/or RSS derived processing parameters file) or proceed with the erroneous parameters by pressing "Execute" button again. In the later case, a warning message will be displayed for confirmation.

If the "Exit" button is pressed, the system prompts the user with the confirmation dialogue. If the user clicks "OK", this window will be closed and this task terminates.

A message line is displayed on the Top Level Dialogue, after the user presses the "Execute" button. The message line indicates the application status and may be one of the following messages:

- "Simulation run in progress, please wait..."
- "Simulation request is denied"
- "Simulation run terminated successfully"
- "Simulation run terminated with errors"
- "Simulation run aborted successfully"
- "Simulation run aborted with errors"



Monitor/Control Simulation Run

Enter Input File Names for Simulation Run:

Simulation Run Definition: /directory/file_name

Signal Data: /directory/file_name

Enter Output Directory Names to store output files generated by this run:

Interm. Results & Detections File Dir.: /directory

Enter Version number for PE software, Number of iteration to playback radar signal, and Run Identifier for this Simulation Run:

PE Software Version:
- PE-1.0
- PE-1.1
- PE-2.0

Number of Iterations: 1

Simulation Identifier: abcdefghijkm

Simulation run terminated successfully.

Validation Message Log

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
line1
line2
line3
line4
line5
line6
line7
line8
line9
line10
```

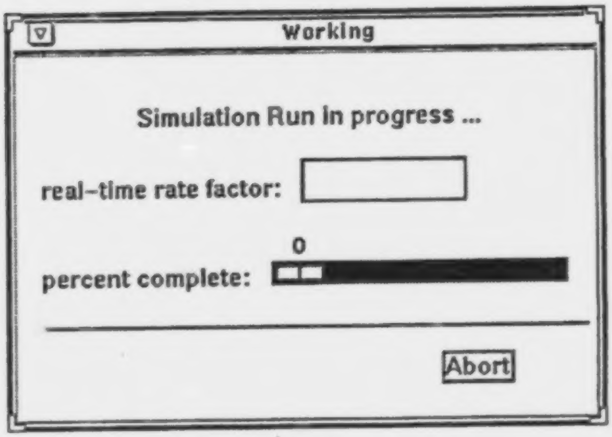
csdhTopLevelD

G6.2 Progress Status Dialogue

This dialogue is displayed while the Simulation Run activity is currently running (busy). The dialogue layout looks similar to the progress status dialogue used by the Monitor/Control Signal Generation task (Refer to G5.2) except that this dialogue will have an extra text field widget for the real-time rate factor display. The scale widget indicates the percentage completion (0% to 100%). Clicking the "Abort" button will terminate the current simulation run activity and return control back to the Top Level Dialogue of Monitor/Control Simulation Run task (See G6.1).

When "Abort" is chosen, a confirmation dialogue box will be presented before Simulation Run is aborted.

While this dialogue is displayed, the top level dialogue input is blocked, and all buttons and fields are disabled.



Simulation Run in progress ...

real-time rate factor:

percent complete: 0

csihWorking

APPENDIX H

RTSG BACKPLANE HARDWARE DESIGN DOCUMENTATION LIBRARY

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

Page

H1 BUILD STANDARD LIST H-1

28-0524 Real Time Signal Generator 3 pages

H2 PARTS LIST H-5

24-5027-D00 Real Time Signal Generator 3 pages
24-5025-D00 VASP IOP Adapter Subassembly 1 page

H3 DRAWINGS H-11

28-0494 Real-Time Signal Generator 16 pages
28-0495 Real-Time Signal Generator 2 pages
28-0561 Real-Time Signal Generator 1 page
28-0449 IOP Adapter Board 1 page
28-0446 IOP Adapter Subassembly 1 page

APPENDIX H

RTSG BACKPLANE HARDWARE DESIGN DOCUMENTATION LIBRARY

H1 BUILD STANDARD LIST

LIBP360

MDA
BUILD STANDARD LISTS

10:39:07 30-AUG-1993

BSL DOCUMENT NUMBER: 28-0524
BSL REVISION NO: D04
BSL REVISION DATE: 30-aug-1993
PROJECT NUMBER: 2512
PAGE: 1 OF 3

UNIT NAME: REAL TIME SIGNAL GENERATOR
UNIT PART NUMBER: 23-9739-D00
UNIT SERIAL NUMBER: 001
PROJECT NAME: SBR-SPD
CUSTOMER: DREO

SUB-ASSEMBLY :	A01	A02	A03	A04
MDA PART NUMBER :	23-9731-D00	$35770	23-9729-D00	23-9729-D00
TITLE:	REAL TIME SIG. GEN. MOTHERBD	VASP EXTENDER	VASP IOP ADAPTR SUBASSEMBLY	VASP IOP ADAPTR SUBASSEMBLY
PARTS LIST :	24-5027-D00		24-5025-D00	24-5025-D00
SCHEMATIC DRAWING:	28-0494-D00		28-0446-D00	28-0446-D00
SUB-ASSY SERIAL NUMBER:	9306004	EXT-0100	9305001	9305002
ASSEMBLY DRAWING NUMBER:	28-0495-D00		28-0577-D00	28-0577-D00
WIRE LIST /DIAGRAM NUMBER:				
ARTWORK:	28-0496-D00		28-0448-D00	28-0448-D00
ASSEMBLY TYPE:	PCB	PURCHASED PCB	PCB	PCB
LOWER LEVEL SUB-ASSEMBLY:				
MECHANICAL DRAWING NUMBER:	FAB 28-0561-D00		FAB 28-0449-D00	FAB 28-0449-D00
PDL. NO./OTHER DATA:				

Q.A VERIFICATION/DATE: ____________ ____________ ____________ ____________
(* = SUPPLIED IN MANUAL)

```
LIBP360                                MDA                      10:39:07  30-AUG-1993
                               BUILD STANDARD LISTS

BSL DOCUMENT NUMBER: 28-0524                UNIT NAME:           REAL TIME SIGNAL GENERATOR
BSL REVISION NO:     D04                    UNIT PART NUMBER:    23-9739-D00
BSL REVISION DATE:   30-aug-1993            UNIT SERIAL NUMBER:  001
PROJECT NUMBER:      2512                   PROJECT NAME:        SBR-SPD
PAGE:                2 OF 3                 CUSTOMER:            DREO

SUB-ASSEMBLY :               A05

MDA PART NUMBER :            23-9729-D00

TITLE:                       VASP IOP ADAPTR
                             SUBASSEMBLY

PARTS LIST :                 24-5025-D00

SCHEMATIC DRAWING:           28-0446-D00

SUB-ASSY SERIAL NUMBER:      9305006

ASSEMBLY DRAWING NUMBER:     28-0577-D00

WIRE LIST /DIAGRAM NUMBER:

ARTWORK:                     28-0448-D00

ASSEMBLY TYPE:               PCB

LOWER LEVEL SUB-ASSEMBLY:

MECHANICAL DRAWING NUMBER:   FAB 28-0449-D00

PDL. NO./OTHER DATA:

Q.A VERIFICATION/DATE:
(* = SUPPLIED IN MANUAL)     ____________  ____________  ____________  ____________
```

LIBP360

MDA
BUILD STANDARD LISTS

10:39:07 30-AUG-1993

BSL DOCUMENT NUMBER: 28-0524
BSL REVISION NO: D04
BSL REVISION DATE: 30-aug-1993
PROJECT NUMBER: 2512
PAGE: 3 OF 3

UNIT NAME: REAL TIME SIGNAL GENERATOR
UNIT PART NUMBER: 23-9739-D00
UNIT SERIAL NUMBER: 001
PROJECT NAME: SBR-SPD
CUSTOMER: DREO

NOTES

1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.

Q.A VERIFICATION/DATE: ______________ ______________ ______________ ______________
(* = SUPPLIED IN MANUAL)

H2 PARTS LIST

```
                                                                          PLANT CODE: PRODUCTION     REQUESTER: TJE
MFG,RE,020,2     MacDonald Dettwiler Production Database                                              PAGE NO:     1
31-AUG-1993 13:42                                  BILL OF MATERIALS REPORT
                                                   ========================
                                         SORTED BY ASSEMBLY PART NUMBER, COMPONENT CLASS CODE
                                                        AS OF 93/08/31

CLASS CODE:   165
MDA ASSEMBLED HARDWARE
PART: 24-5027-D00
DESC: REAL TIME SIGNAL GENERATOR     UOM: EA  SC: M   REV: 3
                                                                          ST QTY PER  EFFECTIV INACTIVE
MANUFACTURER #      MDA PART #       DESCRIPTION                       SC UM ASSEMBLY  DATE     DATE     REFERENCE INFORMATION
------------------  ---------------  --------------------------------- -- -- -------- -------- -------- ----------------------------

  110 INTEGRATED CIRCUITS
------------------------------------
                                     IC CMOS BUFFER/CLOCK DRIVER       P  EA     4.00 00/00/00 99/99/99 U7-U10
49FCT805SO          35706            NON-INVERTING 20PIN SOIC
                                     IC TTL QUAD 2 INPUT AND GATE      P  EA     1.00 00/00/00 99/99/99 U1
74F08D              35707            14PIN SOIC
                                     IC TTL DUAL J-K FLIP FLOP         P  EA     2.00 00/00/00 99/99/99 U3, U5
74F114D             35709            14PIN SOIC
                                     IC TTL QUAD 2 I/P NAND BUFFER     P  EA     1.00 00/00/00 99/99/99 U4
74F38D              35708            OPEN COLLECTOR 14PIN SOIC
                                     IC TTL DUAL RETRIGGERABLE         P  EA     1.00 00/00/00 99/99/99 U2
74LS123D            35710            MONOSTABLE MULTIVIBRATOR 16PIN
                                     SOIC
                                     IC LOGIC DELAY LINE 14 NS       . P  EA     1.00 00/00/00 99/99/99 U6
SMLDL-TTL-14        35711            14PIN SOIC

  111 CRYSTALS
------------------------------------
                                     CRYSTAL OSC 50MHZ 100PPM 0-70C P  EA     1.00 00/00/00 99/99/99 Y1
JS53R-50.0|MHZ      35712            SURFACE MOUNT

  130 CAPACITORS
------------------------------------
                                     CAP CERAMIC 0.22UF 20% 50V        P  EA     2.00 00/00/00 99/99/99 C124,C127
GMC31Z5U224M50NT    35720            SIZE 1206 SURFACE MOUNT
                                     CAP CERAMIC 0.1UF +80 -20% 50V P  EA   119.00 00/00/00 99/99/99 C1-C54,CN1A-CN60A,C119-C123
MCH315F104ZPN       35716            SIZE 1206 SURFACE MOUNT
                                     CAP TANT 1.5UF 10% 16V CASE A  P  EA     1.00 00/00/00 99/99/99 C118
PCT1.5/16AK         35717            SURFACE MOUNT
                                     CAP TANT 22UF 10% 10V CASE C   P  EA    10.00 00/00/00 99/99/99 C109-C117, C128
PCT22/10CK          35719            SURFACE MOUNT
                                     CAP TANT 6.8UF 10% 10V CASE B  P  EA   112.00 00/00/00 99/99/99 C55-C108,C125,C126,CN1B-CN17B,
PCT6.8/10BK         35718            SURFACE MOUNT                                                CN19B-CN45B,CN48B-CN51B,
                                                                                                  CN53B-CN60B

  140 RESISTOR DEVICES/FUSES
------------------------------------
4816P-002-390       34702            RES NET 39 OHM BUSSED             P  EA    44.00 00/00/00 99/99/99 RN11-RN45,RN49-RN53,RN57-RN60
```

```
MFG,RE,020,2     MacDonald Dettwiler Production Database                          PLANT CODE: PRODUCTION    REQUESTER: TJE
31-AUG-1993 13:42                              BILL OF MATERIALS REPORT                                         PAGE NO:     2
                                               ========================
                                   SORTED BY ASSEMBLY PART NUMBER, COMPONENT CLASS CODE
                                                     AS OF 93/08/31
CLASS CODE:    165
MDA ASSEMBLED HARDWARE
PART: 24-5027-D00
DESC: REAL TIME SIGNAL GENERATOR       UOM: EA  SC: M   REV: 3
                                                                      ST QTY PER  EFFECTIV INACTIVE
MANUFACTURER #      MDA PART #          DESCRIPTION                SC UM ASSEMBLY DATE     DATE     REFERENCE INFORMATION
------------------- ------------------- -------------------------- -- -- -------- -------- -------- -------------------------------
4816P-002-390       Continued :         SURFACE MOUNT SOM-PACK 16PIN
4816P-003-331/471   34701               RES NET 330/470 OHM DUAL-TERM P EA   16.00 00/00/00 99/99/99 RN1-RN10,RN46-RN48,RN54-RN56
                                        SURFACE MOUNT SOM-PACK 16PIN
MCR18F1540          35715               RES 154 OHM 1% 1/8W SIZE 1206 F EA   36.00 00/00/00 99/99/99 RT1,3,5,7,9,11,13,15,17,19,21
                                        SURFACE MOUNT                                              23,25,27,29,31,33,35,37,39,41
                                                                                                   43,45,47,49,51,53,55,57,59,61
                                                                                                   63,65,67,69,71
MCR18F2553          35713               RES 255K OHM 1% 1/8W SIZE 1206 P EA   2.00 00/00/00 99/99/99 R1,R5
                                        SURFACE MOUNT
MCR18F73R2          35714               RES 73.2 OHM 1% 1/8W SIZE 1206 F EA  36.00 00/00/00 99/99/99 RT2,4,6,8,10,12,14,16,18,20,22
                                        SURFACE MOUNT                                              24,26,28,30,32,34,36,38,40,42
                                                                                                   44,46,48,50,52,54,56,58,60,62
                                                                                                   64,68,70,72
RM12F1001CT         35721               RES 1K OHM 1% 1/8W SIZE 1206   P EA   4.00 00/00/00 99/99/99 R2-R4,R6
                                        SURFACE MOUNT

  150 CONNECTORS
---------------------------------------

1-215614-4          35705               CONNECTOR FEMALE 3X32 POS      P EA   2.00 00/00/00 99/99/99 A01J2,A02J2
                                        PRESSFIT WIRE WRAP POST 0.669"
100-370-680-6TT-64  35773               COMPLIANT PIN I/O 0.025 SQUARE P EA  80.00 93/03/25 99/99/99 I2,I3,I4,I19,JC3-JC20,
                                        POST, MANUFACTURE PART # SCP-100-370-680-6TT-64            JCQ3-JCQ20
535032-4            35704               CONNECTOR FEMALE 3X32 POS      P EA   2.00 00/00/00 99/99/99 A01J1,A02J1
                                        PRESSFIT WIRE WRAP POST 0.25"
535074-1            35760               SHROUD STANDARD SIZE 0.125"PCB P EA   2.00 00/00/00 99/99/99 A01J2,A02J2
                                        0.521 POST,W/ MOUNTING HOLES
70203-111           35559               SHROUD BACKPANEL CONNECTOR 2MM P EA  16.00 00/00/00 99/99/99 A03D,A04D,A19D,A20D (PIN25-48)
70234-131           35723               HEADER MALE PRESSFIT 4X24 POS  P EA   4.00 00/00/00 99/99/99 A03D,A04D,A19D,A20D (PIN25-48)
                                        0.614" SOLDER TAIL
70234-911           35722               HEADER MALE SIGNAL 4X24 POS    P EA  68.00 00/00/00 99/99/99 A03B-A20B,A05D-A18D (PIN 1-48)
                                        2MM PRESSFIT 0.17" SOLDER TAIL                             A03D,A04D,A19D,A20D (PIN 1-24)
70236-901           35739               HEADER MALE POWER 4X6 POS 2MM  P EA  54.00 00/00/00 99/99/99 A03A-A20A,A03C-A20C,A03E-A20E
                                        PRESSFIT 0.17" SOLDER TAIL
                                        GXT PLATING
70274-001           35751               HEADER CODING PART BLACK       P EA  15.00 93/08/31 99/99/99
MJ-D-1-2            35725               MINI-JUMPER SHORTING CONNECTOR P EA  40.00 00/00/00 99/99/99 I2,I3,I4,I19,JC3-JC20,
                                        BLACK GOLD FLASH PLATED WITH                               JCQ3-JCQ20
                                        PULL TAB

  176 FASTENERS
---------------------------------------

60589               28025               SPACER, HEX, #4-40 X 3/4"      P EA   4.00 93/08/31 99/99/99 IOP - ADAPTER
KF2-256-ET          35724               FASTENER PRESS NUT #2-56       P EA  10.00 00/00/00 99/99/99 P19,P20,P21A,P21B,P21S,P23,
                                        CARBON STEEL                                               P26-P29
KF2-440-ET          04818               PRESS NUT #4-40, 0.112         P EA  37.00 00/00/00 99/99/99 P21,C,D,E,F,G,H,I,J,K,L,M,N,O,
```

H-7

H-8

```
                                                                              PLANT CODE: PRODUCTION    REQUESTER: TJE
MFG,RE,020,2    MacDonald Dettwiler Production Database                                                  PAGE NO:    3
31-AUG-1993 13:42                               BILL OF MATERIALS REPORT
                                                ========================
                                    SORTED BY ASSEMBLY PART NUMBER, COMPONENT CLASS CODE
                                                  AS OF 93/08/31

CLASS CODE:   165
MDA ASSEMBLED HARDWARE
PART: 24-5027-D00
DESC: REAL TIME SIGNAL GENERATOR     UOM: EA  SC: M   REV: 3
                                                                   ST QTY PER  EFFECTIV INACTIVE
                                     DESCRIPTION                SC UM ASSEMBLY DATE     DATE     REFERENCE INFORMATION
MANUFACTURER #     MDA PART #        ------------------------- -- -- -------- -------- -------- --------------------------
------------------ ----------------- CARBON STEEL                                                 P,Q,R,P22A-M,P25A-H
KF2-440-ET         Continued :       SCREW, #4-40 X 1/2" BINDER HD P EA     4.00 93/08/31 99/99/99 IOP - ADAPTER
MS-168             12932             SLOTTED, ZINC PLATED
                                     LOCKWASHER #4-40 INTERNAL    P EA     8.00 93/08/31 99/99/99 IOP ADAPTER
MW-416             06935             ZINC PLATED

  193 ARTWORK OR FILM
------------------------------------
                                     A/W REAL TIME SIGNAL GENERATOR M EA     1.00 93/04/05 99/99/99 ARTFWORKS
UNASSIGNED         28-0496-R00

  NOTES
------------------------------------
```

```
MFG,RE,020,2    MacDonald Dettwiler Production Database                                PLANT CODE: PRODUCTION    REQUESTER: TJE
31-AUG-1993 13:42                                  BILL OF MATERIALS REPORT                                          PAGE NO:    1
                                                   ========================
                                      SORTED BY ASSEMBLY PART NUMBER, COMPONENT CLASS CODE
                                                        AS OF 93/08/31

CLASS CODE:   165
MDA ASSEMBLED HARDWARE
PART: 24-5025-D00
DESC: VASP IOP ADAPTER SUBASSEMBLY    UOM: EA  SC: E   REV: 3
                                                                        ST QTY PER  EFFECTIV INACTIVE
MANUFACTURER #     MDA PART #         DESCRIPTION                    SC UM ASSEMBLY DATE     DATE     REFERENCE INFORMATION
------------------ ------------------ ------------------------------ -- -- -------- -------- -------- ------------------------------

  150 CONNECTORS
--------------------------------------

609-5027           35688              HEADER PCB 50 PIN MALE LOW PRO P  EA     2.00 00/00/00 99/99/99 J2, J3
                                      WITHOUT MOUNTING PROVISON
                                      STRAIGHT CONTACTS
70229-101          35672              CONN MODULAR FEMALE 4X24 POS   P  EA     1.00 00/00/00 99/99/99 J1
                                      RIGHT ANGLE PCB MOUNT 2.00mm CENTERLINE
75406-003          30734              WIREWRAP PINS FOR MACHINE      P  EA     1.00 00/00/00 99/99/99 J4, J5
                                      GXT/NICKEL 8.89MM HIGH PINS, .635MM DIAM.

  160 PRINTED CIRCUIT BOARDS
--------------------------------------

28-0448-R00        35755              IOP ADAPTER PCB                M  EA     1.00 00/00/00 99/99/99

  176 FASTENERS
--------------------------------------

B-1518             05116              NUT, HEX, #2-56                P  EA     2.00 93/08/31 99/99/99
MS-1047            05118              SCREW, #2-56 X 3/8", BINDER HD P  EA     2.00 93/08/31 99/99/99
                                       ZINC PLATED
MS-167             04506              SCREW, #4-40 X 3/8" BINDER HD  P  EA     1.00 93/08/31 99/99/99
                                      SLOTTED, ZINC PLATED
MW-401             06938              WASHER, FLAT, #4, ZINC PLATED  P  EA     1.00 93/08/31 99/99/99
W-306              06088              LOCKWASHER, #2, INTERNAL TOOTH P  EA     4.00 93/08/31 99/99/99
W-309              04618              WASHER SPLIT LOCK #4,.04"WIDE, P  EA     1.00 93/08/31 99/99/99
                                      .031"THICK ZINC PLATED

  999 UNCLASSIFIED
--------------------------------------

UNASSIGNED         28-0614            VASP IOP ADAPTER SUBASSEMBLY   M  EA     1.00 93/08/31 99/99/99

  NOTES
--------------------------------------
```

H-9

THIS PAGE INTENTIONALLY LEFT BLANK

H3 DRAWINGS

FOLD-OUT DIAGRAM 28-0494 pages H-12 to H-27
28-0495 pages H-28, H-29
28-0561 page H30
28-0449 page H31
REMOVED PRIOR TO SCANNING - ? page H32

EXTRA COPIES STORED UNDER SYSTEM NUMBER

153394

APPENDIX I

PARAMETER ESTIMATION FUNCTION

By: Dr. Tim J. Nohara
Raytheon Canada Limited

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

Page

APPENDIX I **PARAMETER ESTIMATION FUNCTION** I-1

I1 Introduction I-1

I2 Pseudo Code Descriptions I-4

APPENDIX I

PARAMETER ESTIMATION FUNCTION

I1 INTRODUCTION

This appendix describes the Testbed function (module) **ComputeParameterEstimates** which is called offline from within PV-Wave in order to process detection results computed from a RUN of the SBR SPD. This function is part of the Data Analysis Task (i.e. the CDAT CSU) described in Section 3.2.5.1 and specifically satisfies the Parameter Estimation requirement (paragraph 3.3.4.0.3 from the Testbed Specification).

This module operates on a single DSP RUN file (i.e. a detection data file) which can include up to twelve NCI data blocks. The DSP detection data file and associated DSP parameters (these include the parameters required by the Parameter Estimation functions) will be made available in the PV-Wave environment as variables. The Parameter Estimation functions will simply operate in the PV-Wave environment on this data in order to produce~~s~~ unambiguous range estimates, angle estimates (including monopulse), RCS estimates and velocity estimates. The output estimates will be written to one or more ascii data files where each record in the file lists the estimates associated with each detection.

In Figure I-1, the sub-modules called by the **ComputeParameterEstimates** module are shown. ExtractSignalProcessingParameters provides a user interface in PV-Wave and reads into PV-Wave a DSP parameter file containing all of the DSP parameters for a given RUN. FormatCurrentDetectionData reads in a single dwell interval of raw detections and formats this data for use by the ComputeRangeEstimates module. The

ComputeRangeEstimates module processes this data and computes an unambiguous detection list. This detection list drives the ComputeRcsEstimates, ComputeMonopulseEstimates, and ComputeVelocityEstimates modules which calculate RCS, angle, and velocity estimates, respectively, for each detection. Finally, the computed estimates are written to file by OutputEstimates .

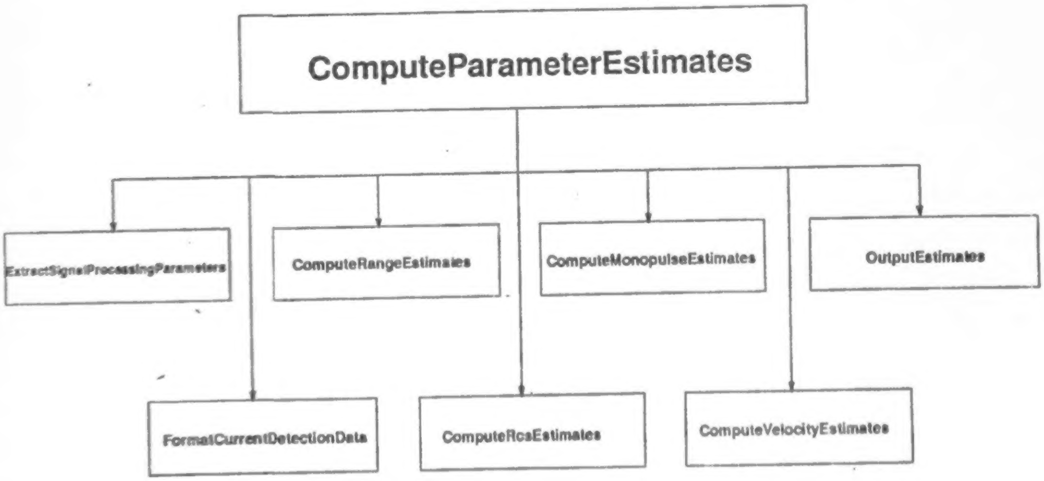

ComputeParameterEstimates
ExtractSignalProcessingParameters
ComputeRangeEstimates
ComputeMonopulseEstimates
OutputEstimates
FormatCurrentDetectionData
ComputeRcsEstimates
ComputeVelocityEstimates

Figure I-1: Module Hierarchy for ComputeParameterEstimates.

The pseudo code description of the **ComputeParameterEstimates** module is provided in Section I2 where the following conventions are observed. Function names are always bolded. In a function definition, we also underline and italicize the function name for clarity. Variable names are italicized. Where ambiguities may arise, a variable that is an array (vector) will be denoted by a single underline below its name; the variable name without the underline refers to an element in the array. The operators .* , ./ , .+, and .- are used with vectors (matrices) and imply element-by-element operations. The first element in array $\underline{u}$ is assumed to be $\underline{u}(1)$. Software library functions are indicated in upper-case, Roman font (not italicized); eg. SIN, COS etc. The symbol $\underline{u}^2$ means each element is squared. Any text which occurs after a semi-colon in the METHOD section of a function definition is for narrative purposes only.

I2 PSEUDO CODE DESCRIPTIONS

ComputeParameterEstimates

SUB-MODULES: ExtractSignalProcessingParameters, FormatCurrentDetectionData ComputeRangeEstimates, ComputeRcsEstimates, ComputeMonopulseEstimates, ComputeVelocityEstimates, OutputEstimates

INPUTS

None.

OUTPUTS

(*n_udet(DI), estimates_xx(UDET)*):

n_udet(DI): -if *dwell_num* = 0, *n_udet(DI)* is an array containing the number of unambiguous detections computed for each of the *n_di* DIs; otherwise, *n_udet(DI)* is a scalar representing the number of unambiguous detections computed for DI-number *dwell_num* .

estimates_xx(UDET):
-if *dwell_num* = 0, then *xx* represents the DI numbers {01,02, ... ,*n_di*} so that *n_di* vectors (arrays) are returned by the function, where each vector, *estimates_xx(UDET)*, contains the *n_udet(xx)* unambiguous detections for DI-number *xx* ; otherwise, a single vector, *estimates_xx(UDET)*, for DI-number *xx* is returned where *xx = dwell_num* .

The elemental structure of the variable *estimates_xx* is as follows:

estimates_xx(UDET).nhits : number of NCIs that contributed hits to the detection
estimates_xx(UDET).hiti: vector (yes=1,no=0) indicating which NCIs had hits
estimates_xx(UDET).tar: list of theoretical ambiguous ranges for each NCI
estimates_xx(UDET).r : unambiguous range estimate (km)
estimates_xx(UDET).v: velocity estimate (m/s)
estimates_xx(UDET).rcs: RCS estimate m^2
estimates_xx(UDET).theta: depression angle estimate (rad)
estimates_xx(UDET).phi: azimuth angle estimate (rad)

This data is left in the PV-Wave environment for the user to do with as he pleases. He can easily store this data to disk by correctly answering the appropriate prompts, or by using PV-Wave command language.

METHOD:

```
BEGIN
        ; Include interface file "cdat_dspir_intf.pro" from MDA so that ingest functions work.
        parameters = ExtractSignalProcessingParameters(xdr_file, param_file, RAR_prfs,dwell_num,
                                                                   out_file);
        prod_handle = DSPIR_OPEN(unit=1, xdr_file) ;             open detection file

        IF (dwell_num = 0) THEN
                first_dwell = 1;   last_dwell = n_di;
        ELSE
                first_dwell = last_dwell = dwell_num;
        END
        FOR ii = first_dwell TO last_dwell DO
                formatted_detections(VEL)(NCI) =
                FormatCurrentDetectionData(prod_handle,ii, parameters);

                RAR_prfs~ = RAR_prfs(ii);      get RAR detection threshold for current dwell
                (n_udet(ii), mono_angle(UDET) ,  estimates_ii(UDET).nhits , estimates_ii(UDET).hiti ,
                estimates_ii(UDET).tar , estimates_ii(UDET).r , estimates_ii(UDET).rcs ,
                estimates_ii(UDET).v     ) =
                ComputeRangeEstimates( formatted_detections, parameters,RAR_prfs~,ii );

                ( estimates_ii(UDET).rcs ) = ComputeRcsEstimates(
                n_udet(ii) , estimates_ii(UDET).rcs , estimates_ii(UDET).r , parameters, ii );

                ( estimates_ii(UDET).theta, estimates_ii(UDET).phi ) =
                 ComputeMonopulseEstimates(n_udet(ii), estimates_ii(UDET).r,
                  mono_angle(UDET), parameters, ii);

                estimates_ii(UDET).v  =
                ComputeVelocityEstimates(n_udet(ii), estimates_ii(UDET).v, parameters,ii  );

                OutputEstimates(
                n_udet(ii), estimates_ii(UDET), xdr_file, param_file, RAR_prfs~, ii, out_file );

        END        ; DO LOOP END
        DSPIR_CLOSE( prod_handle );   close the input detection data file
END
```

ExtractSignalProcessingParameters

SUB-MODULES: DSPPARAM_OPEN, DSPPARAM_RD, DSPPARAM_CLOSE

INPUTS

(*xdr_file, param_file, RAR_prfs, dwell_num, out_file*):

xdr_file : -full pathname of XDR (eXternal Data Representation) detection data file from the RSP for which parameter estimates are to be computed.
param_file: -DSP parameter file that is used with *xdr_file*
RAR_prfs: -vector of RAR detection thresholds, one for each NCI.
dwell_num : -dwell number within *xdr_file* for which parameter estimates are required. The dwell number is bounded between 1 and the number of dwells in the file *xdr_file* ; *dwell_num* = 0 means process all dwells in the file.
out_file: -the root name of the output parameter estimation file.

OUTPUTS

parameters: -a dsp parameter structure as declared in PV-Wave include file cdat_dspparm_intf.pro containing all DSP parameters for the given RUN associated with the detection file *xdr_file*. See also file cdsi_intf.x in the Testbed Detailed Design Document for similar declarations of the parameter structure.

Note: the ComputeParameterEstimates function will access the following parameters from this DSP parameter structure:

PRF(DI)(NCI) : -matrix of PRFs for each DI and NCI.
n_di: -number of DIs in the data file.
n_nci(DI): -number of NCIs per DI.
f_s: -sampling rate of the data (Hz) .
R_ref (DI): -array containing the reference range for range bin zero at beam centre (km), for each DI.
fpl(DI): -array containing the footprint length (km) associated with valid detections, for each DI.
h: -SBR altitude (km) .
θ*(DI)*: -array containing the look-boresight depression angle (rad) for each DI .
ϕ*(DI)*: -array containing the look-boresight azimuth scan angle (rad) for each DI.
rcs_scale(DI): -array containing the RCS scale factor for each DI .
N_pd (DI): -array containing the pulse Doppler length for each DI .
PRF_0 : -Nominal reference pulse repetition frequency (Hz) .
f_0: -Nominal reference carrier frequency (Hz) .
augment_pd(DI): -augment Doppler switch (PC/NCI/none) .
PRF_u(DI): -unambiguous PRF (Hz) .
monopulse_enable : -*monopulse_enable* = 1 signifies that monopulse angle estimates are available.
PD_enable: -*PD_enable* != 1 implies to bypass the pulse Doppler function.
pulse_compr_enable : -*pulse_compr_enable* != 1 implies bypass pulse compression.

METHOD

```
;It is expected that a point-and-click entry screen will be provided to the user for inputting the
;input parameters required, and validity checking will be performed on these inputs.

BEGIN
        ;Include interface file "cdat_dspparm_intf.pro" from MDA for parameter read functions.
        ;determine the DSP parameter file name , param_file,  that corresponds to xdr_file
        ;if it is not available from xdr_file, input it from the user.
        ;Also, provide a GUI to input the parameter RAR_prfs which is an integer scalar if only
        ;a single DI is selected or which is a vector of integers if dwell_num=0 implying all
        ;dwells.  It would be nice also to automatically fill the vector with a constant,
        ;intelligent choice for this case. e.g. the a fraction like 3/4 of the total.
        ;Note: in the parameters RAR_prfs(DI) input in  the Simulation Run HMI are no longer
        ;used and are replaced by the corresponding parameters that are input by this GUI.

         lun = 2;           logical unit number  (1 is used for detection data)
        prod_handle = DSPPARAM_OPEN(lun, param_file);    MDA function to open file
        parameters = DSPPARM_RD(prod_handle  );     MDA function to read in parameters
        DSPPARM_CLOSE(prod_handle  );       MDA function to close parameter file

        ;check for pathological cases in the data
        PD_enable = parameters.radar_proc.PD_enable ;
        pulse_compr_enable = parameters.radar_proc.pulse_compr_enable ;
        IF ( NOT PD_enable  ) THEN
                PRINT("Fatal Error: Invalid data, no velocity bins, ... aborting");
                ABORT;             abort ComputeParameterEstimates
        END
        IF ( NOT pulse_compr_enable  ) THEN
                PRINT("Warning: Operating with uncompressed data");
        END
END
```

DSPPARM_OPEN

INPUTS

(*unit, param_file*):

unit :	-logical unit number for file.
param_file :	-full pathname of XDR (eXternal Data Representation) DSP parameter file.

OUTPUTS

prod_handle :	-file handle pointing to opened file *param_file* .

METHOD

;This function is provided by MacDonald Dettwiler Associates (Stephen Tran) . See
;Section D1.7 of the Testbed Detailed Design Document.

DSPPARAM_RD

INPUTS

(prod_handle):

prod_handle: -pointer to the XDR (eXternal Data Representation) DSP parameter file.

OUTPUTS

parameters:

-a structure in PV-WAVE containing the entire DSP parameter definition for the RUN. All instances of all DSP parameters are represented. The structure of this variable is described in the Testbed Detailed Design Document under cdsi_intf.x .

METHOD

;This function is provided by MacDonald Dettwiler Associates (Stephen Tran).
;See Section D1.7 of the Testbed Detailed Design Document.

DSPPARAM_CLOSE

INPUTS

(*prod_handle*):

prod_handle: -pointer to the XDR (eXternal Data Representation) DSP parameter file.

OUTPUTS

none.

METHOD

;This function is provided by MacDonald Dettwiler Associates (Stephen Tran) . See
;Section D1.7 of the Testbed Detailed Design Document.

FormatCurrentDetectionData

SUB-MODULES: RoundUpEven, DSPIR_RD_NCI

INPUTS

(*prod_handle, dwell_num~ , parameters*):
prod_handle : -pointer to the XDR (eXternal Data Representation) detection data file from the RSP for which parameter estimates are to be computed.
dwell_num~ : -current dwell number (local variable) .
parameters : -parameter vector as defined in **ExtractSignalProcessingParameters** .

OUTPUTS

formatted_detections(UVEL)(NCI) :
THE ELEMENTAL STRUCTURE OF *formatted_detections* IS AS FOLLOWS:
-number of ambiguous ranges for current velocity bin and NCI:
formatted_detections(UVEL)(NCI).n_ambig
-velocity bin number corresponding to each unique velocity represented by *UVEL:*
formatted_detections(UVEL)(NCI).vbin
-array of ambiguous ranges:
formatted_detections(UVEL)(NCI).ranges(1..n_ambig)
-array of monopulse angle information:
formatted_detections(UVEL)(NCI).angles(1..n_ambig)
-array of integrated squared magnitudes :
formatted_detections(UVEL)(NCI).sqmag(1..3)(1..n_ambig)
;Note: *UVEL* means over the number of unique velocity bins in the DI that have
;detections associated with them.

METHOD

;This function reads raw detection data using an MDA-provided function called DSPIR_RD_NCI.
;We must ensure that we can detect a file with no detections. i.e. the calculation of *n_det* within
;this program must return a 'zero' if there are no-detections in the file. Otherwise, the value
;assigned to *formatted_detections(vv)(nci).n_ambig* will be incorrect and this error will propagate.

```
BEGIN
    ;Initializations
    size = SIZE(parameters.DI(dwell_num~).NCI );          PV-Wave SIZE function
    n_nci~ = size(1);                                   n_nci(dwell_num~)
```

```
;get augment_pd(dwell_num~) , the switch for augmenting Doppler bins
augment_pd~ = parameters.DI(dwell_num~).PD.doppler_aug_sel;
N_pd ~ = parameters.DI(dwell_num~).PD.PD_FFT_len; N_pd (dwell_num~)
PRF_u~ =  parameters.DI(dwell_num~).PD.unambig_PRF;  PRF_u(dwell_num~)
;get array of PRFS for each NCI in current DI, ie.PRF(dwell_num~)(1...n_nci~)
PRFs =  parameters.DI(dwell_num~).NCI(1..n_nci~).PRF;
monopulse_enable = parameters.general_config.monopulse_enable;

IF (augment_pd ~ = 'none') THEN
        nvbins = N_pd ~;
ELSE
        N_a = RoundUpEven(   N_pd ~* (  PRF_u~  /  MIN( PRFs  )   -  1)  );
        nvbins = N_pd ~+ N_a  ;
END

;scan through detection data to determine number of unique velociy bins and to create a
;cross map between unique bin index and its correponding velocity bin value.
vcount = ZEROS(nvbins, 1) ;          column vector of nvbins zeros (1...nvbins)
FOR jj = 1 TO  n_nci~  DO
        ; retrieve from file the current NCI's detections
         detections(1 ... n_det)  = DSPIR_RD_NCI(prod_handle, dwell_num~, jj);
         size = SIZE( detections );    PV-WAVE function
        n_det = size(1);    NOTE: we must ensure n_det = 0 if no detections in file.

        FOR kk = 1 TO n_det DO
            vel = detections(kk).v ;    velocities between 0 and N_a-1
            vcount(vel + 1) = 1;        mark the occurance of this velocity
        END         ; end of detections DO loop
END       ; end of nci DO loop
unique_velocities = FIND(vcount = 1) - 1;  set of unique  velocities bin numbers
vmap = ZEROS(nvbins, 1) ;            cross-reference map for unique velocities
nuvbins =  LENGTH(unique_velocities) ; number of unique velocity bins
vindex = 1 : 1 : nuvbins ;    unique velocity indices
vmap(unique_velocities + 1) = vindex ;     set cross-references

;Initialize formatted_detections matrix
FOR vv = 1 TO nuvbins DO,  FOR jj = 1 TO  n_nci~  DO
        formatted_detections(vv)(jj).n_ambig = 0;
        formatted_detections(vv)(jj).ranges(1) = 0 ;
        IF (monopulse_enable) THEN
            formatted_detections(vv)(jj).angles(1) = 0 ;
        END
```

```
            formatted_detections(vv)(jj).sqmag(1)(index) = 0 ;
            formatted_detections(vv)(jj).sqmag(2)(index) = 0 ;
            formatted_detections(vv)(jj).sqmag(3)(index) = 0 ;
        END, END

        ;read in detection data and format
        FOR jj = 1 TO  n_nci~  DO
            ; retrieve from file the current NCI's detections
             detections(1 ... n_det)  = DSPIR_RD_NCI(prod_handle, dwell_num~, jj);
             size = SIZE( detections );    PV-WAVE function
            n_det = size(1);    NOTE: we must ensure n_det = 0 if no detections in file.

            ; arrange detections into formatted_detections
            vcount = ZEROS(nvbins, 1) ;                   column vector of nvbins zeros
            FOR kk = 1 TO n_det DO
                vel = detections(kk).v ;
                index = ++vcount(vel);            increment detection counter
                formatted_detections(vmap(vel+1))(jj).n_ambig = index ;
                formatted_detections(vmap(vel+1))(jj).vbin = vel;
                formatted_detections(vmap(vel+1))(jj).ranges(index) = detections(kk).r ;
                IF (monopulse_enable) THEN
                    formatted_detections(vmap(vel+1))(jj).angles(index) = detections(kk).a ;
                END
                formatted_detections(vmap(vel+1))(jj).sqmag(1)(index) = detections(kk).p(1) ;
                formatted_detections(vmap(vel+1))(jj).sqmag(2)(index) = detections(kk).p(2) ;
                formatted_detections(vmap(vel+1))(jj).sqmag(3)(index) = detections(kk).p(3) ;
            END        ; end of detections DO loop
        END       ; end of nci DO loop
END   ;end of FormatCurrentDetectionData
```

DSPIR_OPEN

INPUTS

(*unit, xdr_file*):

unit :	-logical unit number for file.
xdr_file :	-full pathname of XDR (eXternal Data Representation) detection data file from the RSP for which parameter estimates are to be computed.

OUTPUTS

prod_handle :	-file handle pointing to opened file *xdr_file* .

METHOD

;This function is provided by MacDonald Dettwiler Associates (Stephen Tran) . See
;Section D1.7 of the Testbed Detailed Design Document.

DSPIR RD NCI

INPUTS

(*prod_handle*, *dwell_num~*, *nci_num*):

prod_handle:	-pointer to the XDR (eXternal Data Representation) detection data file from the RSP for which parameter estimates are to be computed.
dwell_num~ :	-dwell number of update interval to extract (local variable).
nci_num :	-NCI number of update interval to extract (local variable).

OUTPUTS

detections(1 ... n_det):

-a vector of detection data corressponding to DI-number *dwell_num~* and NCI-number *nci_num* from the detection data file pointed to by *prod_handle* . That is, the returned vector corresponds to all of the ambiguous-range detections listed in the detection data file for the specified NCI and DI . The detection elements are ordered both in velocity bin number and range bin number; the outer ordering is according to increasing velocity bin number (0 to *N_pd* + *N_a* -1) and the inner ordering (most rapidly changing) is according to range bin number (0 to *N_r*-1). A detection element is a structure containing the following components:

.v the velocity bin number,
.r the range bin number,
.a the monopulse angle information (if present),
.p(3) integrated target power array of length 3, ie. left_adjacent, target, and right_adjacent powers.

For example, *detection(11).p(2)* refers to the integrated target power in the target bin (*.v*, *.r*) for the 11th detection element in the specified DI-number and NCI-number .

METHOD

;This function is provided by MacDonald Dettwiler Associates (Stephen Tran).
;See Section D1.7 of the Testbed Detailed Design Document.

DSPIR_CLOSE

INPUTS

(prod_handle):

prod_handle:	-pointer to the XDR (eXternal Data Representation) detection data file from the RSP for which parameter estimates are to be computed.

OUTPUTS

none.

METHOD

;This function is provided by MacDonald Dettwiler Associates (Stephen Tran) . See
;Section D1.7 of the Testbed Detailed Design Document.

RoundUpEven

INPUTS

real_number : -real number scalar

OUTPUTS

even_integer : -the smallest even integer number that is greater than or equal to *real_number* .

METHOD

```
BEGIN
    even_integer = CEIL(real_number) ;        currently the next largest integer
    IF ( MOD(even_integer,2) != 0 ) THEN
        even_integer = even_integer + 1;
    END
END
```

ComputeRangeEstimates

SUB-MODULES: ComputeFootprintLimits

INPUTS

(formatted_detections, parameters,RAR_prfs~, current_dwell):
formatted_detections :
-formatted detection vector containing one DI of the following:
formatted_detections(UVEL)(NCI).n_ambig
formatted_detections(UVEL)(NCI).vbin
formatted_detections(UVEL)(NCI).ranges(1..n_ambig)
formatted_detections(UVEL)(NCI).angles(1..n_ambig)
formatted_detections(UVEL)(NCI).sqmag(1..3)(1..n_ambig)
parameters : -parameter vector containing the following elements:
PRF(DI)(NCI) , n_di, n_nci(DI), RAR_prfs(DI), f_s , R_ref(DI) , fpl(DI), h, θ(DI), φ(DI), rcs_scale(DI), N_pd(DI) , PRF_0 , f_0 , augment_pd(DI) , PRF_u(DI), monopulse_enable .
RAR_prfs~ : RAR detection threshold for current dwell (i.e. the M in M out of N)
current_dwell : -current dwell number for use with the parameter vector

Note: see functions **ExtractSignalProcessingParameters** and **FormatCurrentDetectionData** for more information on the inputs .

OUTPUTS

(n_udet~, mono_angle(1...n_udet~), estimates(1... n_udet~).nhits, estimates(1...n_udet~).hiti, estimates(1...n_udet~).tar, estimates(1...n_udet~).r, estimates(1...n_udet~).rcs , estimates(1...n_udet~).v) :

Note: see function **ComputeParameterEstimates** for a complete description of these quantities. These variables are defined for the current dwell number, *current_dwell* .

METHOD

;This routine implements an M out of N RAR algorithm to estimate unambiguous range.
;Ambiguous detections are replicated over the entire valid footprint; if at least M of the
;NCIs detect a target in a given unambiguous range bin, then a detection is declared; otherwise,
;it is ignored. Since up to 12 NCIs can be defined, M and N are less than or equal to 12.
;This algorithm does not use the Chinese Remainder Theorem. Therefore, there are no
;restrictions placed on the PRFs that are used. The RAR is based on the m-file RAR.M
;written by Peter Weber.

```
BEGIN
        ;Constants used below
        lneg  = -1000;     large negative number  (mono_angle > lneg always)
        rlneg = lneg + 1;  used to account for floating point inaccuracy of lneg
        sol = 3e5;         speed of light (km/s)

        ;Initializations are done below
        size = SIZE( formatted_detections) ;        use the PV-Wave SIZE command
        nvbins = size(1);                  number of unique velocity bins in formatted_detections
        nprf = size(2);                    number of PRFs (NCIs) in current dwell
        size = SIZE(parameters.DI(current_dwell).NCI ); PV-Wave SIZE function
        n_nci~ = size(1);                              n_nci(current_dwell)
        IF ( n_nci~ != nprf ) THEN
                PRINT('fatal error: detection data file and DSP parameters file incompatible');
                EXIT;          exit ComputeParameterEstimates
        END
        total_n_udet = 0;      total number of unambiguous detections for the dwell
        n_udet~ = 0;           initialize the number of output detections for this dwell
        ;get  R_ref(current_dwell) , the current zero-range reference (km)
        R_ref~ =  parameters.DI(current_dwell).engag_geom.ref_range;
        ;get  fpl(current_dwell) the current valid footprint length (km)
        fpl~ =  parameters.DI(current_dwell).parm_est.footprint_len;
        ;get array of PRFS for each NCI in current DI, ie.PRF(current_dwell)(1...nprf)
        PRFs =  parameters.DI(current_dwell).NCI(1..nprf).PRF;
        f_s = parameters.sampling_rate ;          sampling rate (Hz)
        pris = f_s * ONES(1,nprf ) ./ PRFs(1 ... nprf ) ;   vector of PRIs
        h = parameters.altitude ;                 altitude (km)
        monopulse_enable = parameters.general_config.monopulse_enable;

        ;Compute defined valid footprint limits
        (fp_start,  fp_end)  =
        ComputeFootprintLimits(R_ref~, fpl~, h  );

        ;Main RAR loop begins below
        FOR vv = 1 TO nvbins DO
                ;Extract current velocity bin data for RAR.  Ensure that alist, plist, anglist and
                ;nd are properly defined, even if there are no detections for a given prf (nd(nn)=0).
                current_velocity_bin = formatted_detections(vv)(1).vbin ;
                FOR nn = 1 TO nprf DO
                    alist(nn,:) = formatted_detections(vv)(nn).ranges ;   2D range matrix
                    IF (monopulse_enable) THEN
        anglist(nn,:) = formatted_detections(vv)(nn).angles ;   2D angle matrix
      END
      plist(nn,:) = formatted_detections(vv)(nn).sqmag ;    3D power matrix
      nd(nn) = formatted_detections(vv)(nn).n_ambig ;       1D array
    END   ;nci DO loop end

    ;Compute unambiguous range estimates
    fp_len = fp_end - fp_start + 1;          footprint length (bins)
    maxpri = MAX( pris(1:nprf) );            largest PRI (bins)
    udet = ZEROS(1, fp_len);                 initialize integer unambiguous detection vector
    adet = ZEROS(nprf, maxpri+2);            integer ambig. det. matrix
    pdet = ZEROS(nprf, maxpri+2);            power matrix for RCS
    mdet = lneg * ONES(nprf, maxpri+2);monopulse matrix initialized
    ;Note: above three matrices are 2 larger than necessary to have placeholders for
    ;range straddling which can occur to the left or right of a given range bin. i.e. bin
    ;1 is placed at location 2 so that straddling to the left of bin 1 can be placed at
    ;location 1 in the matrices.  In reality, straddling to the left from bin 1 implies bin
    ;PRI, but this requires a MOD operation which is expensive inside the two
    ;loops below.  Similarly, bin PRI is stored in location PRI+1 and straddling to
    ;the right is indicated in location PRI+2.
    FOR p = 1 TO nprf DO
      FOR d = 1 TO nd(p) DO
        ;algorithm below assumes ambiguous range between 1 and pris(p) .
        ar = alist(p, d) + 1;                get current ambig. range and make ≥ 1
        adet(p, ar+1) = 1;                   adet is set to 1 at ambiguous detection ranges
        pdet(p, ar+1) = plist(p, 2,d);          pdet is set to the power of bin ar
        IF (monopulse_enable) THEN
          mdet(p, ar+1) = anglist(p, d);        mdet is set to the angle
        END
        ;adjacent ranges with strong enough power also set to account for
        ;range bin straddling.  The updates on pdet appear to result with the
        ;last update being the survivor.  This doesn't matter since all updates
        ;to the same location are updated with the same value by definition of plist.
        ;The updates to mdet result in the angle corresponding to the nearest
        ;detection bin being replicated in the straddling bin locations.  If there is a
        ;bin straddled between two detection bins, then the angle assigned to that
        ;bin is from the first detection that accesses that straddled bin.
        IF (plist(p, 1,d) > 0.7 * plist(p, 2,d) ) THEN
          adet(p, ar) = 1;                   set left straddling bin
          pdet(p, ar) = plist(p, 1,d);  set power to power of left straddling bin
          IF (mdet(p, ar) < rlneg AND monopulse_enable) THEN
            mdet(p, ar) = anglist(p, d);  set angle to that of detected bin
            END
        END

        IF ( plist(p, 3,d) > 0.7 * plist(p, 2,d) ) THEN
            adet(p, ar+2) = 1;         set right straddling bin
            pdet(p, ar+2) = plist(p, 3,d); set power to that of right straddling bin
            IF (mdet(p, ar+2) < rlneg AND monopulse_enable ) THEN
                mdet(p, ar+2) = anglist(p, d);  set angle to that of detected bin
            END
        END
    END        ;end of ambiguous detection loop for given PRF

    ;resolve the first-last range wrap around:
    adet(p, 2) = MAX( adet(p, 2), adet(p, pris(p)+2) );    if either set, then set
    pdet(p, 2) = MAX(pdet(p, 2), pdet(p, pris(p)+2));      use largest power
    IF ( mdet(p, 2) < rlneg AND monopulse_enable) THEN
        mdet(p, 2) = mdet(p, pris(p)+2);   use PRI+2 angle if not already valid
    END

    ;resolve the last-first range wrap around:
    adet(p, pris(p)+1) = MAX(adet(p, 1), adet(p, pris(p)+1));      set if either
    pdet(p, pris(p)+1) = MAX(pdet(p, 1), pdet(p, pris(p)+1));      use largest power
    IF (mdet(p, pris(p)+1) < rlneg AND monopulse_enable ) THEN
        mdet(p, pris(p)+1) = mdet(p, 1);      use 1 angle if not already valid
    END

    ;Replicate ambiguous detections across the footprint and add at each range
    rr = fp_start:1:fp_end;   vector of unambiguous ranges in footprint
    ambr = MOD(rr-1, pris(p)) + 1;      vector of ambiguous ranges
    udet = udet + adet(p, ambr+1);  accumulate detections for current PRF
END   ;end of p=1:nprf Loop

;Determine which unambiguous ranges have at least M out of N hits,
;i.e. RAR_prfs~ out of nprf
r = FIND(udet ≥ RAR_prfs~ ) + fp_start - 1;   list of detection range bins
n_udet~ = LENGTH(r);        number of unambig. detections (output) for this vel.

;Assemble output quantities corresponding to each unambigous detection
ci = total_n_udet + 1 : 1 : total_n_udet + n_udet~  ; current indices
mono_angle(ci) = ZEROS(1,n_udet~);        initialize monopulse angles
FOR d = ci  DO
    avg_rcs = 0;        initialize average rcs over NCIs
```

```
                pmax = 0;          initialize maximum power
                FOR p = 1 TO nprf DO
                    ar = MOD(r(d-total_n_udet)-1, pris(p)) + 1; theoretical ambig. range bin
                    tar(p) = ar;  store theoretical ambiguous range bin number for output
                    hiti(p) = adet(p, ar+1);  store hit indicator  (yes = 1, no = 0)
                    avg_rcs = avg_rcs + pdet(p, ar+1);  mean rcs incremented by power
                    ;assign monopulse angle quantity
                    IF (pdet(p, ar+1) > pmax AND monopulse_enable) THEN
                        pmax = pdet(p, ar+1);   store new maximum power
                        mono_angle(d) = mdet(p, ar+1);angle corresponds to maximum power
                    END
                END   ;end of PRF loop for computing average rcs
                nhits = SUM(hiti);     no. of PRIs that contributed to this unambig. detection
                avg_rcs = avg_rcs / nhits;        average rcs over PRIs which contributed
                estimates(d).nhits = nhits;       number of hits for current detection
                estimates(d).hiti = hiti ;        hit indicators versus PRF (i.e. NCI)
                estimates(d).tar = tar;           list of theoretical ambiguous ranges
                estimates(d).r = r(d-total_n_udet)*sol/(2*f_s);unambig. range estimate (km)
                estimates(d).rcs = avg_rcs ;      average rcs
                estimates(d).v = current_velocity_bin;          velocity bin
            END  ;end of unambiguous detection loop
            total_n_udet = total_n_udet + n_udet~ ;    accumulate total number of detections
        END ;end of main velocity bin DO loop
        n_udet~ = total_n_udet;   total unambiguous detections for the current dwell
END  ;end of ComputeRangeEstimates
```

ComputeFootprintLimits

INPUTS

(R_ref~ , fpl~, h):

R_ref~ : -reference range (km) for the current dwell.

fpl~ : -footprint length (km) for valid targets, for current dwell.

h : -altitude (km) for the current run.

OUTPUTS

(fp_start, fp_end):

fp_start: -the start of the valid footprint in terms of the integer number of range bins from the SBR.

fp_end: -the end of the valid footprint in terms of the integer number of range bins from the SBR.

METHOD

```
BEGIN
    ;All radar distances are relative to the SBR
    sol = 3e5 ;          speed of light in km /s
    Re = 6371.0 ;        radius of the earth (km)
    fp_start = MAX(h , R_ref~ - fpl~ / 2 );   start of valid, user-defined footprint (km)
    horizon = SQRT( (Re + h )^2 - Re^2 );  distance to the horizon (km)
    fp_end = MIN(horizon , R_ref~ + fpl~ / 2 );  end of valid, user-defined footprint
    fp_start = (2 f_s ) * fp_start / sol ;        convert from (km) to range bins from SBR
    fp_end = (2 f_s ) * fp_end / sol;             convert from (km) to range bins from SBR
END    ; end of ComputeFootprintLimits
```

ComputeRcsEstimates

INPUTS

(n_udet~ , rcs(1..n_udet~) , uranges(1..n_udet~) , parameters, current_dwell):

n_udet~:	-the number of detection (unambiguous range) entries output from the ComputeRangeEstimates function for the current DI.
rcs(1..n_udet~):	-average squared magnitude (A/D units) for each output detection entry from the ComputeRangeEstimates function.
uranges(1..n_udet~):	-unambiguous range (km) for each output detection entry from the ComputeRangeEstimates function.
parameters:	-the parameter vector from the function ExtractSignalProcessingParameters .
current_dwell :	-the current dwell number in the RUN used to index parameters within the parameter structure *parameters*.

OUTPUTS

rcs_m2(1..n_udet~) :	-the radar cross section in units m^2 corresponding to each squared magnitude entry in the input vector *rcs(1..n_udet~)* .

METHOD

```
BEGIN
        ;initializations
        ;get rcs_scale(current_dwell)  the rcs scale factor for current dwell
        rcs_scale~ = parameters.DI(current_dwell).parm_est.RCS_scale_factor ;

        ;compute target rcs estimates in units of (m^2)
        rcs_m2(1..n_udet~) = ( rcs(1..n_udet~) .* ( uranges(1..n_udet~).^4 ) ) * rcs_scale~ ;
END     ;end of ComputeRcsEstimates function
```

ComputeMonopulseEstimates

INPUTS

(*n_udet~*, *uranges(1..n_udet~)*, *mono_angle(1...n_udet~)* , *parameters*, *current_dwell*):

n_udet~: -the number of detection (unambiguous range) entries output from the ComputeRangeEstimates function for the current DI.

uranges(1..n_udet~): -unambiguous range (km) for each output detection entry from the ComputeRangeEstimates function.

mono_angle(1...n_udet~): -array of angle quantities used to determine the monopulse angle estimate for each detection if *monopulse_enable* == 1; otherwise, just a dummy vector of zeroes.

parameters: -the parameter vector from the function ExtractSignalProcessingParameters .

current_dwell : -the current dwell number in the RUN used to index parameters within the parameter structure *parameters*.

OUTPUTS

(*theta_estimates(1..n_udet~)*, *phi_estimates(1..n_udet~)*):

theta_estimates(1..n_udet~): -the look-down angle (radians) to each of the targets associated with the *uranges* vector. If monopulse is available, then high resolution monopulse estimates are returned; otherwise, the nominal beam pointing angle is returned.

phi_estimates(1..n_udet~): -the azimuth-scan angle (radians) to each of the targets associated with the *uranges* vector. If monopulse is available, then high resolution monopulse estimates are returned; otherwise, the nominal beam pointing angle is returned.

METHOD

```
BEGIN
    ;required parameters for this function
    Re = 6371.0;                average radius of earth (km)
    ;get $\theta$(current_dwell) the look-down angle (rad) for current dwell
    theta~ = parameters.DI(current_dwell).engag_geom.look_depression;
    ;get $\phi$(current_dwell) the azimuth scan angle for current dwell (rad)
    phi~ = parameters.DI(current_dwell).engag_geom.look_azimuth;
    h = parameters.altitude ;                altitude (km)
    monopulse_enable = parameters.general_config.monopulse_enable;
```

```
        IF (monopulse_enable) THEN
            sin_beta_l = COS(theta~ ) * COS(phi~ );   look-angle monopulse reference
            tmp1 = FIND(ABS( mono_angle(1...n_udet~) + sin_beta_l  ) ≤ 1 );
            tmp2 = FIND(ABS( mono_angle(1...n_udet~) + sin_beta_l  ) > 1 );
            sin_beta(tmp1) =  mono_angle(tmp1) + sin_beta_l ;
            sin_beta(tmp2) =  sin_beta_l  * ONES(1, LENGTH(tmp2 ) );
            theta_estimates(1..n_udet~) = ASIN( (h^2+uranges(1..n_udet~).^2 + 2*h*Re)  /
                                                ( 2*(h+Re)*uranges(1..n_udet~)  ) );
            phi_estimates(1..n_udet~) = ACOS(sin_beta ./ COS(theta_estimates)  );
        ELSE
            theta_estimates(1..n_udet~) = theta~ * ONES(1,n_udet~) ;
            phi_estimates(1..n_udet~) = phi~ * ONES(1,n_udet~);
        END
END  ;end of ComputeMonopulseEstimates
```

ComputeVelocityEstimates

INPUTS

(*n_udet~, vel(1..n_udet~), parameters, current_dwell*);

n_udet~:	-the number of detection (unambiguous range) entries output from the ComputeRangeEstimates function for the current DI.
vel(1..n_udet~):	-velocity bin number for each output detection entry from the ComputeRangeEstimates function.
parameters:	-the parameter vector from the function ExtractSignalProcessingParameters .
current_dwell :	-the current dwell number in the RUN used to index parameters within the parameter structure *parameters*.

OUTPUTS

velocity(1.. n_udet~):-output vector of velocities (m/s) corresponding to each velocity bin entry in the input vector *vel* .

METHOD

;Note: if time and money are available, it would be possible to correct these velocity estimates ;using the high resolution monopulse angles. Initially, the value of *velocity(ii)* lies between 0 and *Nv* - 1 (see definition of *Nv* below)

```
BEGIN
    ;extract parameters for current dwell
    sol = 3e8;                     speed of light in m/s
    ;get N_pd(current_dwell) the number of pulse doppler bins for current dwell
    Npd~ = parameters.DI(current_dwell).PD.PD_FFT_len ;
    PRFu = parameters.DI(current_dwell).PD.unambig_PRF;  unambig. PRF (Hz)
    augment_pd~ = parameters.DI(current_dwell).PD.doppler_aug_sel;
    size = SIZE( parameters.DI(current_dwell).NCI  );  number of NCIs
    n_nci~ = size(1);
    PRFs =  parameters.DI(current_dwell).NCI(1...n_nci~).PRF;
    PRF_0  = parameters.radar_proc.nominal_PRF;  nominal PRF (Hz)
    f_0 = parameters.radar_proc.nominal_carrier_freq; nominal frequency (Hz)
```

```
        ;compute the number of velocity bins at the input to the CFAR detector
        Na = 0;
        IF ( augment_pd~ != 'none' ) THEN
                Na = RoundUpEven(Npd~ * ( PRFu / MIN(PRFs ) - 1 ) ); no. augmented cells
        END
        Nv = Npd~ + Na ; total number of velocity bins

        ;make velocity bins bipolar
        FOR ii=1 TO n_udet~ DO
                IF ( vel(ii) ≥ Nv / 2 ) THEN
                    vel(ii) = vel(ii) - Nv ; negative velocities
                END
        END

        ;convert velocity bin number to m/s
        velocity(1.. n_udet~) = vel(1..n_udet~) * ( PRF_0 * sol / (2 * Npd~ * f_0 ) );
END     ;end of ComputeVelocityEstimates
```

OutputEstimates

INPUTS

(*n_udet~, estimates_xx(1..n_udet~), xdr_file, param_file,RAR_prfs~, current_dwell, out_file*);

n_udet~:	-the number of detection (unambiguous range) entries output from the ComputeRangeEstimates function for the current DI.
estimates_xx(1..n_udet~):	-the estimation output structure defined for each of the *n_udet~* detections . The format of this structure is as described in function ComputeParameterEstimates .
xdr_file :	-full pathname of XDR (eXternal Data Representation) detection data file from the RSP for which parameter estimates are to be computed.
param_file:	-DSP parameter file used with *xdr_file*
RAR_prfs~:	-number of NCIs with hits for an unambiguous detection
current_dwell :	-current dwell number within *xdr_file* for which parameter estimates have been computed and will be written to file.
out_file:	-the root name of the output parameter estimation file.

OUTPUTS

None. Only an output file is created if requested by user . See the function ExtractSignalProcessingParameters . One file per dwell is create storing the detections for that dwell. For a given name, the output file name is made up of a root filename plus the dwell number.

METHOD

```
BEGIN
    ;initializations and writing out of header information
    lun = 3;                              logical unit number, 1 and 2 are already used
    output_filename = out_file + "_" + NUM2STR( ii ) ;      file name for current dwell
    FOPEN(lun, output_filename);              open the formatted output file (ASCII file)
    FWRITE(lun, xdr_file);                    write XDR file name
    FWRITE(lun, param_file);                  -write DSP parameter file name
    FWRITE(lun, current_dwell);               write XDR file dwell number
    FWRITE(lun, RAR_prfs~);                   write  RAR_prfs~ for current DI
```

```
        ;write detection data to file
        FOR jj = 1 TO n_udet~ DO
                FWRITE(lun, estimates_xx(jj).nhits, estimates_xx(jj).hiti, estimates_xx(jj).tar,
                estimates_xx(jj).r,  estimates_xx(jj).v, estimates_xx(jj).rcs,
                estimates_xx(jj).theta, estimates_xx(jj).phi );
        END
        FCLOSE(lun);                            close the output file
END
```

APPENDIX J

RSS ALGORITHMS

By: P. Weber,
Raytheon Canada Limited

Dr. T. Nohara,
Raytheon Canada Limited

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

Page

APPENDIX J RSS ALGORITHMS J-1

J1 Introduction J-2

J2 Computing and Accessing RRA Pettern J-2

J2.1 RRA Pettern Computation Algorithm J-2

J3 Computing and Accessing CFA Patterns J-3

J3.1 Internal Pattern Tables J-3
J3.2 External Pattern Tables J-4
J3.3 Accessing CFA Pattern Tables J-4

J4 RSS Algorithmic Flow J-4

J4.1 Interpolation of Earth Clutter Signal J-5
J4.2 Interpolation Algorithm J-5

J5 Generating Noise Signals J-6

J5.1 Noise Covariance Matrix Computation Algorithm J-6
J5.2 Noise Signal Generation Algorithm J-6

J6 Generating Dispersed Noise-Jammer Signals J-7

J6.1 Antenna Array Dispersion J-7
J6.2 Large Aperture Dispersion J-7
J6.3 Multipath Dispersion J-8
J6.4 Jammer Power Spectrum J-9

J7 Jammer Types and Characteristics J-9

J8 Generating Jammer Signals J-11

- J8.1 Barrage, Swept, or Follower Jammer Signal Generation Algorithm J-11
- J8.2 Deception Jammer Signal Generation Algorithm J-12

J9 IF Filters J-12

- J9.1 IF Filter and Decimate Algorithm J-12
- J9.2 IF Filter Response Computation Algorithm J-13

FIGURES

J-1 RSS Algorithmic Flow J-5

APPENDIX J

RRS ALGORITHM

J1 Introduction

The algorithms for the RSS were described in Appendix J of the Testbed Preliminary Design Document [R-1]. This appendix will describe some of these in more depth.

J2 Computing and Accessing RRA Pattern

Computing the RRA pattern is done during run initialization. With circularly symmetric aperture shading functions, the error-free unscanned pattern is also circularly symmetric about boresight. This pattern is therefore a one-dimensional function of $\sqrt{u^2 + v^2}$. u and v are direction cosines of position, with RRA axis being the w-axis (i.e. on axis, $u = v = 0$). The pattern is the Hankel transform of the radial shading function; Hankel transform algorithms are not conveniently available. Therefore, RRA patterns are computed as the Fourier transform of the equivalent linear aperture function, which is the integral of the two-dimensional shading function along one axis.

With electronic scanning, u_0 and v_0 are u and v at beam centre. The pattern is modelled to be circularly symmetric about the point (u_0, v_0). With phase errors due to aperture imperfections, circular symmetry no longer holds. Random perturbations in the pattern are modelled as the sum of an error function of u and another function of v. This leads to significant savings in storage and computation, while still having realistic local variation of sidelobes. Thus anytime the gain of the reflector is required, the following implementation is used:

$$\text{rra\_gain}(u,v) = \text{main\_pattern}\left(\sqrt{(u-u_0)^2 + (v-v_0)^2}\right) + \text{u\_error\_pattern}(u-u_0) + \text{v\_error\_pattern}(v-v_0)$$

Each of the three patterns are stored as a table. The RRA pattern is generated once, assuming a nominal RF frequency. Every burst, the indexing of the tables is re-scaled according to the actual RF (the pattern contracts, and therefore Δu decreases, with increasing frequency).

J2.1 RRA Pattern Computation Algorithm

```
for each $x$ point along aperture
    compute blockage and aperture edge $y$-limits
    for each $y$ point between these limits
        compute aperture_voltage (function of $x^2 + y^2$)
        add to equivalent_linear_voltage (function of $x$)
    add equivalent_linear_voltage$^2$ to input_power
(offset fed reflector can be modelled by setting blockage to zero)
Fourier transform equivalent_linear_voltage
scale transform by $\pi D/\lambda$ and input_power$^{-1}$ to get error_free_pattern
compute feed_pattern as per error_free_pattern (different aperture_voltage and D)
```

```
for angular regions beyond the subreflector shadow
    add feed_pattern to error_free_pattern to get main_pattern
generate two random phase error sequences, and apply complex exponential
multiply equivalent_linear_voltage by each complex error sequence to form u_shading and
v_shading sequences
Fourier transform the two shading sequences
subtract error_free_pattern from each transform to form u_error_pattern and
v_error_pattern
```

J3 Computing and Accessing CFA Patterns

Corporate Fed Array antenna patterns are represented by look-up tables which are generated during a burst initialization procedure before each burst of data is generated. The changes in scan angles and carrier frequency which occur each burst are used to update the CFA pattern tables for a given burst. During the signal generation of the burst, these tables are accessed repeatedly in order to provide antenna gains to individual scatterers. Therefore, the burden of the pattern calculation has been placed in the burst initialization procedure for efficiency.

Up two seven CFA antenna apertures can be defined: the transmit aperture, and up to three receive sub-apertures for each of two DPCA groups. Each aperture pattern is represented by at least two look-up tables, one for each principal axis pattern *u* and *v*. In some cases, the antenna elements and array factor are represented separately by their principal axis patterns, thereby requiring four look-up tables to completely describe the overall aperture pattern.

Two different mechanisms are provided for the computation of CFA patterns: internally generated patterns and externally generated patterns. Internally-generated antenna patterns are computed by the RSS with no outside intervention. The user defines all the parameters necessary to describe the antenna in the Satellite and Radar HMI, and these parameters are all used in the computation of the antenna patterns. Typical parameters are the number and types of elements in each array dimension, the aperture shading, the grid pattern, et cetera. A mechanism is also provided for the import of externally-generated patterns. DND has a program called PHASAR which generates complete phased array antenna patterns and models them to great detail. These complete (i.e. element patterns and array factors are already combined) antenna patterns can be imported into the RSS, and formatted into two look-up tables, one for each principal axis pattern *u* and *v* so that they can be accessed during signal generation in a similar manner as the internally-generated look-up tables.

J3.1 Internal Pattern Tables

Computation of internally-generated pattern tables involves the computation of the array factor and the element pattern for each antenna. The array factor computation differs depending on whether the elements for a rectangular grid pattern or a triangular (hexagonal) grid pattern on the face of the antenna. Triangular grids are computed as the sum of two rectangular grids, one offset diagonally from the other.

For each rectangular grid, the aperture shading or illumination function, the antenna steering function, and a random error function used for simulating random sidelobes are represented at each element location along the two principal axes, *u* and *v*; the two complex sequences are individually Fourier transformed to arrive at the principal axis patterns. Grating lobes are then modelled in the Fourier pattern domain. Uniform, Taylor, and Hamming shading functions are supported.

Four different element types are supported and result in the generation of none, one or two principal axis tables. The omni element requires no table by definition as its pattern is not directional. The Bolt-Cassen and dipole elements require only one table to represent their patterns due to pattern symmetry; the Bolt-Cassen table is accessed by $u^2 + v^2$ and the dipole pattern is simply a function of *u*. The horn however requires both *u* and *v* tables to properly represent its pattern.

J3.2 External Pattern Tables

Generation of look-up tables from externally-generated CFA patterns requires reading in the data from the pattern file that corresponds to the two principal axes, $u=0$ and $v=0$. Since the external pattern file is represented in a normalized *U,V* space (i.e. scanning angles and carrier frequency normalized out), a transformation is used to map between the two. The external file is read at the beginning of each burst since scanning and carrier frequency variations will map different vectors in the external pattern file onto the principal axes *u* and *v*. The user who provides the external pattern file must ensure that the domain of the external file is sufficient to cover the range of scanning angles and carrier frequencies established in the HMI for the run.

J3.3 Accessing CFA Pattern Tables

During signal generation, CFA look-up tables will be accessed by an antenna identifier and the direction cosines, *u* and *v*, for each scatterer location for each sampling instant in burst. The overall gain is computed as the product of the principal axes gains $g1(u)$ and $g2(v)$ for externally-generated patterns, and as the product of the element gain ($g1_e(u) * g2_e(v)$ in general) and the array factor gain ($g1_a(u) * g2_a(v)$) for internally-generated patterns.

J4 RSS Algorithmic Flow

Figure J-1 illustrates the high-level flow of the signal generation and radar receiver models. Earth clutter signals are generated first, and then interpolated to a higher sampling rate. Other signals are then generated and added at the higher rate. The front-end non-linearity is then applied, followed by the band-limiting IF filter. At this point, the signals are reduced to the final sampling rate, baseband errors are modelled and the A/D model is applied.

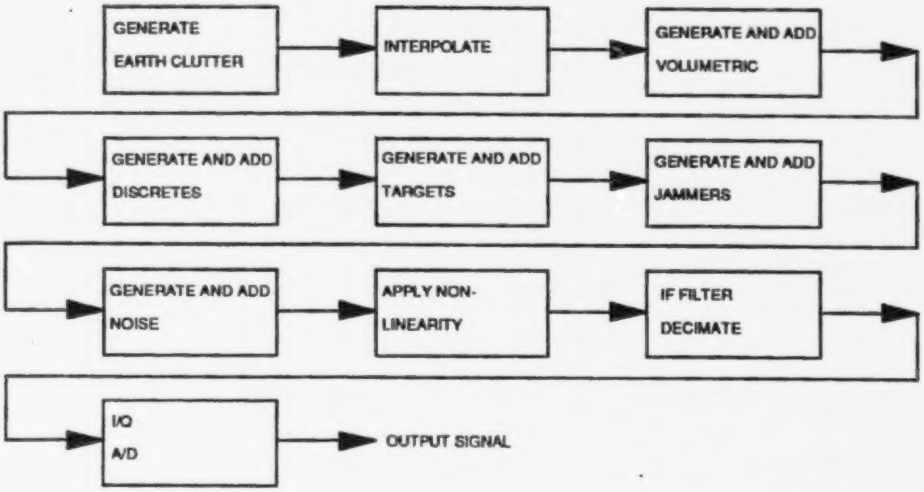

GENERATE
EARTH CLUTTER
INTERPOLATE
GENERATE AND ADD
VOLUMETRIC
GENERATE AND ADD
DISCRETES
GENERATE AND ADD
TARGETS
GENERATE AND ADD
JAMMERS
GENERATE AND ADD
NOISE
APPLY NON-
LINEARITY
IF FILTER
DECIMATE
I/Q
A/D
OUTPUT SIGNAL

Figure J-1: RSS Algorithmic Flow

J4.1 Interpolation of Earth clutter Signal

The RSS has the ability to generate its signals at higher sampling rates than that of the radar system, for more accuracy. In order to save computation time, earth clutter signals can be generated at a lower rate than noise and jammer signals. The jammer/noise oversampling factor is the ratio of the sampling rate for signals other than earth clutter and the A/D output signal. The clutter oversampling factor is a similar ratio for the earth clutter signal. These ratios are user-selected from 1, 2, 4, 8, 16, 32, with the constraint that the other signals are generated at a greater (or equal) rate than (to) the earth clutter signals. There is thus a need for an interpolation function that re-samples the earth clutter signal at the higher rate.

J4.2 Interpolation Algorithm

```
applied to each channel independently
for each time segment (segments have overlap)
    Fourier transform input sequence
    multiply by interpolation window
    inverse Fourier transform the product (zero-padded)
    extract output segment samples (non-overlapped portion only)
(zero-padding factor is equal to the ratio of the oversampling factors)
```

J5 Generating Noise Signals

The noise signals in different channels are correlated for two reasons:

1) delay taps from the same aperture have correlated noise because of its finite bandwidth
2) array apertures have correlated noise if they share elements

The second effect will compute the correct correlation for external noise, which is modelled as independent for disjoint apertures.

To incorporate correlation, the antenna-to-antenna covariances are computed and formed into a matrix. Noise is then generated by:

1) generating a matrix of independent noise signals
2) filtering the signals with the square root of the noise covariance matrix
3) applying delays to the filtered signals to get the various tap signals

The algorithms are summarized below.

J5.1 Noise Covariance Matrix Computation Algorithm

```
for each antenna (sub-apertures of CFA are different antennas in this context)
    enter its noise_power on diagonal of noise_covariance matrix
    for each antenna of higher index
        compute antenna-antenna correlation coefficient using dimensions, locations, grid
        spacings, scan angles, etc.
        scale it by the geometric mean of the two noise_powers and enter into
        noise_covariance_matrix
compute Cholesky_factor of noise_covariance_matrix
```

J5.2 Noise Signal Generation Algorithm

```
for each antenna
    for each time segment
        generate white_sequence (non-overlapped portion) and concatenate with
        overlapped portion
        Fourier transform to white_spectrum
        multiply white_spectrum by noise_window to get noise_spectrum
        for each channel of higher index
            multiply noise_spectrum by complex exponential (this applies tap delay)
            inverse Fourier transform the product
            muliply non-overlapped portion of the transform by Cholesky_factor element
            and add to noise_sequence
```

J6 Generating Dispersed Noise-Jammer Signals

Define $J_{pc}(t)$ as the signal from jammer p in channel c as a function of continuous time t. The total signal in channel c is:

$$A_c(t) = \sum_{p=1}^{njam} J_{pc}(t) + s_c(t) + n_c(t) \qquad \text{(J-1)}$$

the sum of the jammers $J_{pc}(t)$, return signal $s_c(t)$ and noise $n_c(t)$.

For an array composed of antenna elements whose inter-element separation is large relative to their physical size, the dominant dispersion effect is that each antenna receives a jammer signal sample that originated at a slightly different time. This effect is considered first. The dispersion effects on elements with larger apertures are subsequently considered.

J6.1 Antenna Array Dispersion

The distance travelled by the jammer-p wavefront to antenna a is:

$$\rho_{pa} = \rho_p - x_a \sin\theta_p \cos\phi_p - y_a \sin\theta_p \sin\phi_p - z_a \cos\theta_p \qquad \text{(J-2)}$$

where (x_a, y_a, z_a) is the (cartesian) position of the phase centre of antenna a, and $(\rho_p, \theta_p, \phi_p)$ is the (polar) position of jammer p. Both positions are defined in the antenna co-ordinate system. The jammer position is time-varying; the effects of this are addressed later.

The jammer-p arrival-time at antenna a is:

$$T_{pa} = \frac{\rho_{pa}}{c_0} \qquad \text{(J-3)}$$

where c_0 is the speed of light. Each antenna is allowed multiple delay taps, each of which form a radar signal channel. Let c denote the channel number. For a given SBR configuration, there is a mapping from channels $c = 1, \ldots, nchan$ to antennas $a = 1, \ldots, nant$. An example: channels 1 and 2 originate from antenna 1 and channels 3 and 4 originate from antenna 2. The jammer-p signal in channel c is:

$$J_{pc}(t) = G_a(\theta_p, \phi_p)\, J_p(t - T_{pa} - \tau_c) \qquad \text{(J-4)}$$

where $G_a(\theta_p, \phi_p)$ is the voltage gain of antenna a towards jammer p, $J_p(t)$ is the jammer-p signal (with attenuation and propagation losses included), and τ_c is the tap delay applied to channel c.

J6.2 Large Aperture Dispersion

At a given angular position θ_p, ϕ_p, a large aperture is assumed to have a gain versus frequency response derived from its simulated antenna pattern (which is only generated at the centre frequency f_0), as:

$$G_a(\theta_p,\phi_p,f) = G_a\left(\frac{f\sin\theta_p\cos\phi_p}{f_0},\frac{f\sin\theta_p\sin\phi_p}{f_0},f_0\right)$$
$$= G_a\left(\frac{fu_p}{f_0},\frac{fv_p}{f_0},f_0\right) \qquad \text{(J-5)}$$

The jammer signal in such an aperture is therefore:

$$J_{pc}(t) = \mathrm{FT}^{-1}[\ G_a(\theta_p,\phi_p,f)\ \mathrm{FT}[J_p(t-T_{pa}-\tau_c)]\ ] \qquad \text{(J-6)}$$

where the notations FT[] and FT^{-1}[] denote Fourier transformation and its inverse.

J6.3 Multipath Dispersion

The spacecraft carrying SBR can potentially introduce local multipath problems. With multipath, channels receive jammer signals over a number of paths as:

$$J_{pc}(t) = \sum_{q=0}^{npath} J_{pqc}(t)$$
$$J_{p0c}(t) = J_{pc}(t)\ above \qquad \text{(J-7)}$$
$$J_{pqc}(t) = FT^{-1}[\ \alpha_{pqa}(f)\ G_a(\theta_{aq},\phi_{aq},f)\ FT[J_p(t-T_{pqa}-\tau_c)]\ ]$$

npath is the number of multipath scattering centres on the spacecraft; for the qth centre: T_{pqa} is the jammer-*p* arrival time along path *pqa*, (θ_{aq},ϕ_{aq}) is the angle of scattering centre *q* with respect to antenna *a*, $\alpha_{pqa}(f)$ is the reflection coefficient along path *pqa*. Using $G(\theta_{aq},\phi_{aq},f)$ and $\alpha_{pqa}(f)$ is an approximation; both the aperture and the scattering centre are within each other's near field. q = 0 denotes the direct path; $\alpha_{p0a}(f) = 1$, $T_{p0a} = T_{pa}$, and $(\theta_{a0},\phi_{a0}) = (\theta_p,\phi_p)$ when considering jammer-p.

The user specifies *npath*, the number of multipath scatterers, and for each, its position (x_q,y_q,z_q), maximum reflection coefficient γ_q and lobe width β_q. The lobe width parameter ensures that the path-to-path reflection coefficient covariance $E[\alpha^*_{pqa}\alpha_{\chi\xi b}]$ is proportional to the nearness (in both incident and reflected angles) of paths *pqa* and $\chi\xi b$, as:

$$\alpha_{pqa}(f) = \gamma_q\cos\left(\frac{fu_p}{f_0\beta_q}\right)\cos\left(\frac{fv_p}{f_0\beta_q}\right)\cos\left(\frac{fu_{qa}}{f_0\beta_q}\right)\cos\left(\frac{fv_{qa}}{f_0\beta_q}\right) \qquad \text{(J-8)}$$

where (u_{qa},v_{qa}) are the direction cosines of antenna *a* with respect to multipath scatterer *q* and (u_p,v_p) are the boresight direction cosines.

J6.4 Jammer Power Spectrum

The jammer spectra are Gaussian of user-specified 3-dB bandwidth B_p and centre frequency fc_p as:

$$S_p(f) = \frac{ERP_p}{4\pi\rho_p^2 L_p}\exp\left(\frac{-\alpha^2(f-fc_p-fd_p)^2}{B_p^2}\right) \qquad \text{(J-9)}$$

where ERP_p is the effective radiated power of jammer p, ρ_p is its range, fd_p is its Doppler (both re-computed intermittently), L_p is the atmospheric attenuation for jammer p, and $\alpha = 1.665$.

Noise-jammer signals are modelled with normally distributed quadrature components (equivalent to a Rayleigh amplitude distribution).

J7 Jammer Types and Characteristics

The following jammer types are simulated:

J7.1 A barrage jammer transmits a noise signal occupying a fixed band. Barrage jamming is modelled as described above. The user specifies the 3-dB bandwidth B and the centre frequency fc. The signal is generated at baseband frequency $fc-f_0$ only if it overlaps the current pass band ($f_0-f_s/2$ to $f_0+f_s/2$).

J7.2 A swept jammer covers a wide band of frequencies with a narrower band noise-signal by sweeping its carrier frequency at a regular rate. Simulated swept jamming has user-specified sweep rate f_{sw}, swept bandwidth B_{sw} and mid-sweep-frequency f_{mid}. The jammer is first generated as a barrage jammer (using the user-specified instantaneous 3-dB bandwidth B and $fc=0$). It is then determined if the swept signal overlaps the pass band ($f_0-f_s/2$ to $f_0+f_s/2$). If it does, then the jammer signals are multiplied by $\exp(j2\pi(f_i-f_0)n/f_s)$, where f_i is the instantaneous sweep frequency:

$$f_i = f_{mid} - \frac{B_{sw}}{2} + \left\langle \left(\frac{n}{f_s} - t_{start}\right) f_{sw} B_{sw} \right\rangle_{B_{sw}} \qquad \text{(J-10)}$$

where the notation $<a>_b$ means the remainder of a modulo b. If the swept signal is not in-band, then the jammer is set to zero.

J7.3 A follower jammer monitors the SBR frequency and transmits within a narrow band surrounding the monitored frequency. Simulated follower noise jamming has user-specified processing delay T_d (to account for the time it takes the jammer to estimate SBR frequency) and frequency offset fc (to allow a follower that concentrates on a specific Doppler band). The delay is applied after a change in SBR frequency, during which jamming continues at the previous SBR frequency. Referenced to the change-time at SBR,

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g., Establishment sponsoring a contractor's report, or tasking agency, are entered in Section 8.)

MacDonald Dettwiler
13800 Commerce Parkway
Richmond, B.C. V6V 2J3

Raytheon Canada Ltd.
400 Phillip Street
Waterloo, Ont. N2J 4K6

2. **SECURITY CLASSIFICATION** (overall security classification of the document, including special warning terms if applicable)

Unclassified

3. **TITLE** (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title)

Testbed Detailed Design (U)

4. **AUTHORS** (Last name, first name, middle initial)

MacDonald Dettwiler	Anderssen, U. Sawatsky, R. Tran, S.
Raytheon Canada	Collyer, R. Nohara, T. Premji, A. Scarlett, P. Weber, P.

5. **DATE OF PUBLICATION** (month and year of publication of document)

September 1993

6a. **NO. OF PAGES** (total containing information. Include Annexes, Appendices, etc.

1870

6b. **NO. OF REFS** (total cited in document)

19

DESCRIPTIVE NOTES (the category of the document, e.g., technical report, technical note or memorandum. If appropriate, enter the type of report, e.g., interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Testbed Detailed Design Document

8. **SPONSORING ACTIVITY** (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

Space-Based Radar Project, DREO
3701 Carling Avenue
Ottawa, Ontario K1A 0Z4

9a. **PROJECT OR GRANT NO.** (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

Project #: D6471

9b. **CONTRACT NO.** (if appropriate, the applicable number under which the document was written)

Contract No.: W8477-0-TD-08/01-SV
File No.: DREO 3702-194 (SBR)

10a. **ORIGINATOR'S DOCUMENT NUMBER** (the official document number by which the document is identified by the originating activity. This number must be unique to this document)

SB-DS-50-4059

10b. **OTHER DOCUMENT NOS.** (Any other numbers which may be assigned this document either by the originator or by the sponsor)

95-621

11. **DOCUMENT AVAILABILITY** (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

12. **DOCUMENT ANNOUNCEMENT** (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.)

Full announcement.

13. **ABSTRACT** (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

U This document defines and documents the detailed design of the Space-Based Radar (SBR) Signal Processing Demonstrator (SPD) Testbed Subsystem.

The signal processing demonstrator is a laboratory based computer system for the demonstration and validation of SBR signal processing algorithms and real-time digital signal processing hardware. The SPD consists of two subsystems: a Digital Signal Processor (DSP) Subsystem, and Testbed Subsystem.

The Testbed subsystem provides the environment required to operate, test and demonstrate the DSP subsystem. It consists of a Radar Signal Simulator (RSS) and a Testbed Controller. In principle, the Testbed is a complete test instrument capable of being used in conjunction with DSPs other than that developed under this contract. U

14. **KEYWORDS, DESCRIPTORS or IDENTIFIERS** (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g., Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Space-Based Radar (SBR)
Signal Processing Demonstrator (SPD)
Radar Signal Simulator (RSS)
Testbed Controller
Pulse Doppler Radar
Displaced Phase Centre Antenna (DPCA)
Moving Target Indication (MTI)
Radar simulation
Radar environment
Radar clutter models
Radar target models
Radar jammer models
Phased array antenna
Reflector antenna
Monopulse
Human-machine interface
Workstation
Detailed Design

#153394

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM